.15926 Editor

Version 1.4

Volume 1

Getting Started

Welcome to the .15926 Editor. This free software is distributed by TechInvestLab.ru “as is” without any warranties or obligations, for testing purposes. Use it and enjoy at your own risk. Please send bug reports, usage and licensing questions or suggestions to dot15926@gmail.com .

February 23, 2013

Volume 1. Getting Started

Contents

Other documentation volumes:

Volume 2. APIs: Scanner and Builder
Volume 3. Extensions
Volume 4. Patterns and Mapping

License

Parts of .15926 Editor (built-in extensions and extension samples) are released as a source code under the BSD 2-Clause license.

Other parts of the software (released in binary and in text form) are not covered by the license above and are distributed "as is" and free of charge for evaluation purposes only!

Elephant icon by Martin Berube is used for .15926 software according to terms at
http://www.iconarchive.com/show/animal-icons-by-martin-berube/elephant-icon.html

1. Introduction

Welcome to the version 1.4 of **.15926 Editor** (pronounced “dot 15926”). This software application is built on the .**15926 Platform** to demonstrate Platform's capabilities.

.15926 Platform is a name for an architecture and a set of specific interfaces and libraries to work with ISO 15926 data. It is developed by TechInvestLab.ru to facilitate creation of semantic applications to work with ISO 15926 data in all possible ways – read, visualize, explore, search, reason, map, write, exchange, etc.

All efforts are taken for .15926 Platform to be fully compatible with iRING architecture and achieve compliance to JORD requirements. Please refer to the separate .15926 Platform Compliance Report for more details.

In **.15926 Editor** you can:

- Browse ISO 15926 upper ontology in three different namespaces: PCA, RDS/WIP or ISO.
- Search and navigate public ISO 15926 SPARQL endpoints,
  … or any other SPARQL endpoint you like, with authorization if required,
  … including search for legacy RDS/WIP identifiers.
- Search, navigate and edit reference data files distributed publicly, including ISO 15926-4, PCA RDL and ISO 15926-8 templates,
  ... or any other RDF files you like.
- Build complex data project from local files and endpoints, bringing reference data, template definitions and project data together for integrated navigation and verification, customizing namespaces, properties and meta-data attributes.
- Design and run intricate semantic queries or whole data mining and verification algorithms for ISO 15926 data or any other RDF data,
  … using the power of Python general purpose programming language through full-featured REPL environment,
  … and accessing APIs of various .15926 Platform components to read, analyze and change reference and project data.
- Create from scratch your own reference classes and templates, create project data (including template instances) manually or through your own adaptors,
  … in forms ready for file exchange or upload to triple store,
  … generating URI in your namespaces using UUID compliant with RFC 4122 / ITU-T X.667 / ISO/IEC 9834-8.
- Compare data sources, build diff files, review changes and create versioning system for reference and project data, or for any ontology.
- Define data patterns, search for patterns in your data, and visualize search results, map spreadsheets to patterns.
- Extend .15926 Editor functionality (develop your own mapping adapters for example) using Python, any external Python libraries and APIs of .15926 Platform components,
  … testing and debugging them in the .15926 Editor environment,
  … registering them as .15926 Editor extensions,
  … and distributing them as open-source if you like.
- Use or modify open-source extensions from TechInvestLab.ru:
  - conversion of reference and project data from TabLan.15926 data description tables (.xlsx) to ISO 15926 RDF;
  - import of reference data from JSON files created by engineering catalog application (third party);
  - creation or import of template definitions in iRING spreadsheet format.
- Explore (with somewhat limited capabilities) any large RDF datasets,
  … OpenCYC knowledge base, for example.

.15926 Editor is a tool designed with three major goals in mind:

- explore existing sources of reference data in as many formats as possible;
- verify reference data;
- engineer and manage new reference data, including automated reference data creation through adaptors incorporating mapping from external sources.

The Editor is intended to become for ISO 15926 data what Protégé became for OWL data – a primary tool for data exploration.

The Editor is not designed to support any particular reference data management or data integration workflow. Specific applications for this can be built on .15926 Platform tailored to the requirements of organizations exchanging data – namespaces, endpoints, properties, databases, transport layers, etc. Mapping components are integrated in .15926 Platform environment as extensions using external or internal pattern mapping descriptions and directly accessing APIs of source/target databases.

Please contact TechInvestLab.ru at dot15926@gmail.com for further detail on licensing and adaptation of this software.

2. Installing and running the program

Released version works in MS Windows (XP, Vista, 7) only, no other OS versions available.

Download page: http://techinvestlab.ru/dot15926Editor

You can select MS Windows installer or archived version. Download the distribution to your hard drive and follow instructions for installation.

You can also separately download archive with sample data from http://techinvestlab.ru/DataSamples/

In this documentation we'll refer to the folder with software executable as <*installation_folder*>, and to the sample data folder as <*samples*>.

To run the Editor please run *dot15926.exe* from <*installation_folder*> or use icons created.

To avoid loss of your data please don't change sample data files and extensions distributed with the software, or backup them before update and uninstall .

Avoid putting the program or data on virtual drives, external flash drives or network drives, as system may have restrictions to access them. Always test software ability to access folders you have selected for your data.

You can check for updates from the Editor through *Help-Check for a new version* menu command. The Editor tries to access http://techinvestlab.ru and check whether newer version is available. If newer version is found – opening of download page in Internet browser is suggested**.**

3. Glossary

.15926 project (project) – several data sources opened simultaneously in the .15926 Editor, with coherent set of properties (module names, namespaces, object and data properties) defined for each data source.

Active data panel – the data panel in focus, click on its tab or anywhere on visible data panel to make it active.

Annotation property – property that is declared *rdfs:subPropertyOf* of OWL standard annotation properties in ISO 15926-8 or used as standard annotation property in a particular RDL (formally it is also a literal property and in this documentation two sets are often described beside each other).

Configuration of the Editor (configuration) – configuration of the Editor includes location of standard files, location of a current project description file, and locations of Python modules used by the extensions (configuration is kept in the ***15926.cfg*** file in the <*installation_folder*>).

Data item – anything identifiable with URI: data type, reference and project data entity, template definition, template role, property or template role value.

Data panel (panel) – the screen area where data view can be rendered.

Data source (source) - data set opened in .15926 Editor. It can be a file, a group of files, a SPARQL endpoint, or an internal datatype set.

Data source name root node (root node) – a root node representing data source in a data tree and named with the data source name.

Data tree (tree) – an unfoldable tree structure used to organise data in data view.

Data view (view) - information extracted from data source and organized for rendering on screen for exploration and editing (data view may not be actually rendered in any data panel at a given moment).

Data view node (node) – a node in a data tree representing data source, a data entity or a group of entities, or any other data item.

Diff view – information about the difference between two data sources rendered in a data panel for review and approval.

Extension – Python code extending .15926 Editor functionality and integrated in the .15926 Editor as a set of source-code modules following specified rules.

Initial view – for project and reference data source - data view with only the data source name root node exposed, for template definitions data source – data view with all templates exposed.

ISO 15926 - # (**Part #)** – abbreviations used for # -th part of ISO 15926 standard (parts 2, 4, 7 and 8 are referenced in this documentation).

Literal property – property restricted by some XML Schema type.

Module name – unique name assigned to a data source for referencing in APIs of various .15926 Platform components.

Object property (role) – property which has data types or data entities as its range (*rdf:type*, relationship role, or template instance role are examples of object property).

Pattern – in the context of .15929 Platform pattern is a formally defined list of alternative (but compliant to ISO 15926) ways to express particular ontological relation between several entities (Part 7 template axioms are patterns, for example).

Platform component – .15929 Platform software component realizing certain software functionality (sometimes available to users as documented APIs) and distributed in binary (compiled) form.

Project description file – local file with ***.15926*** extension containing .15925 project state (list of data sources and their locations with properties for each data source).

Project panel – the screen area where data sources added to .15926 Editor project and data views are listed.

Reference and project data entity (data entity) – an instance of ISO 15926-2 data type or template instance (in non ISO 15926 data sources data entities are just any subjects in RDF triples).

Relationship – in the context of the Editor interface and in this documentation "relationship" is often used to refer to all instances of relational types of ISO 15926 – to instances of relationships proper, of classes of relationships and classes of classes of relationships, and even to template instances.

Source name – the mnemonic name assigned to data source for convenience of project browsing. The source name is initially assigned by default rules and can be later changed to any other text, not necessarily even unique.

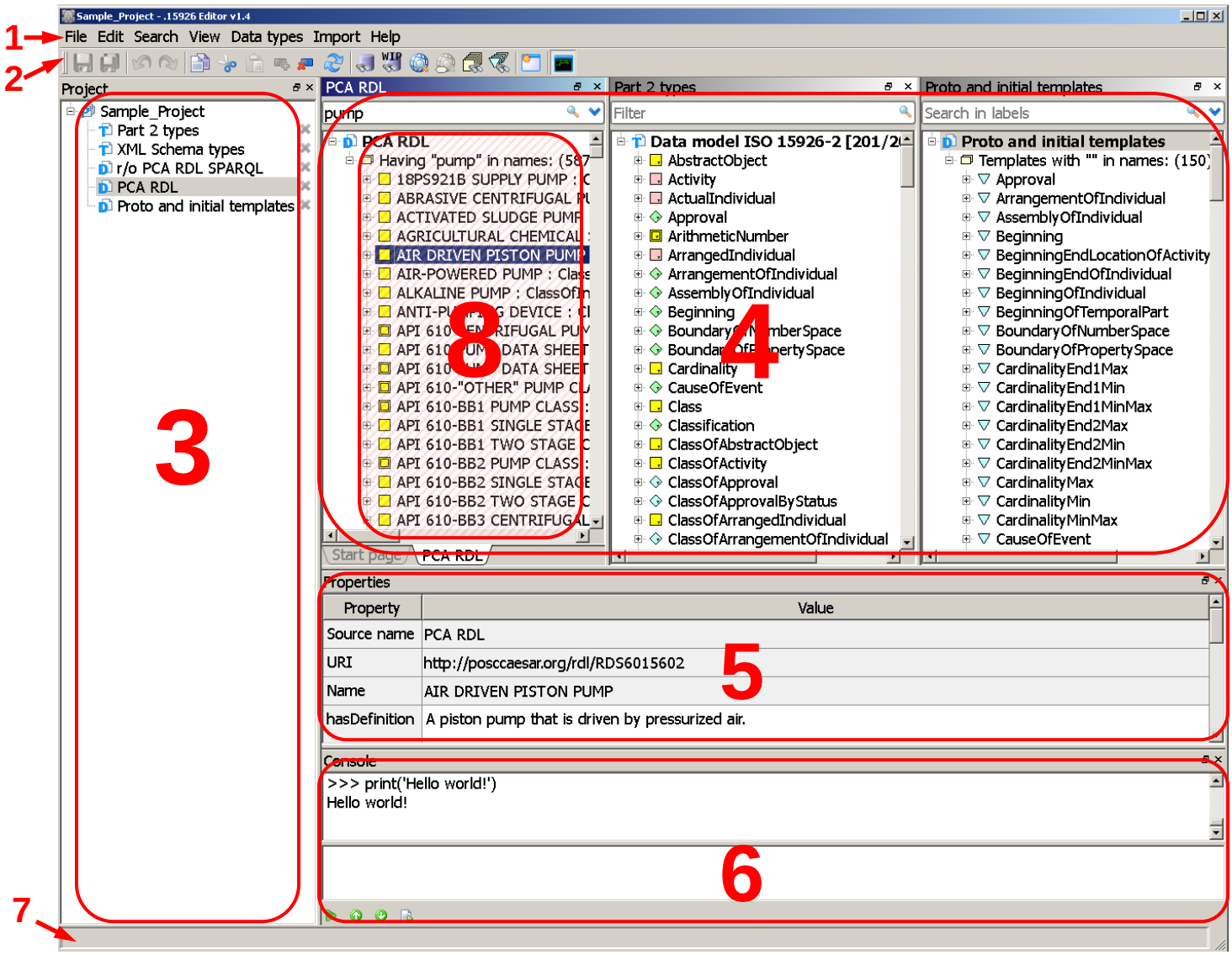
Start page – special panel displayed at the first start of the Editor and available as one of data panels for quick access to recent projects and data sources.

Template definition (template) – RDF graph defining a Part 7 template predicate according to the requirements of Part 8.

4. User interface

4.1. Interface overview

Typical look of .15926 Editor main window (extensions may alter the main menu):



Interface of the .15926 Editor has the following main elements:

1. Main menu
2. Toolbar
3. Project panel
4. Data panels
5. Properties panel
6. Python console
7. Status bar
8. Data view

All panels and Console can be undocked from the main window by clicking the *Undock* button on the title bar of the panel or by dragging the title bar outside of the main window with mouse. Panels are docked again by dragging the title bar inside the main window.

All panels and Console can be rearranged inside the main window. Project panel, Properties panel and Console can be stacked on top of each other in one area on the main window and accessed via tabs. Data panels also can be stacked for tabbed access.

Program's font size can be changed by *View-Font size…* menu commands or by *Ctrl*+= and *Ctrl*+- keyboard shortcuts (increase/decrease) or by *Ctrl+MouseWheel*. Shortcut *Ctrl+0* (*Ctrl-Zero*) restores default font size. Font size adjustments are convenient for public presentations.

4.2. Keyboard shortcut list

Main menu

Alt+Letter - press **Alt** and use underlined letter to access Main menu items and their submenus, use arrows to move in menus

All panels

Ctrl+= – increase font size
Ctrl+- – decrease font size
Ctrl+MouseWheel – increase/decrease font size
Ctrl+0 (Ctrl-Zero) – restore default font size

File

Ctrl+Shift+N – New project…
Ctrl+Shift+O – Open project…
Ctrl+S – Save
Ctrl+Shift+S - Save as...
Ctrl+Alt+S - Save all data sources and current project
Alt+S – Save snapshot…
Ctrl+B – Stop task
Ctrl+Shift+W – Reimport…
Ctrl+Shift+R - Reload modules
Ctrl+Shift+W - Reload patterns
Alt+X - Exit

Edit

Ctrl+Z - Undo
Ctrl+Y – Redo
Ctrl+C – Copy
Ctrl+Shif+C – Copy text
Ctrl+X – Cut
Ctrl+V – Paste
Ctrl+Shif+V – Paste as triples
Insert - Insert new item
Shift+Ins - Add template
Delete - Delete item or node group
F5 - Reload item
Alt+U - Put new UUID onto the clipboard

Search

F4 – Search endpoints for URI
F7 – Search PCA endpoint for WIP equivalent
F6 - Open URI in web browser

View

F12 – Open element in new panel
Ctrl+P – Toggles pattern identification
Ctrl+L – Toggles simplified entity view
Ctrl+` - Toggles Python console
Ctrl+T – Toggles Project panel
Ctrl+G – Toggles Properties panel

Python console

Up – Line up
Down – Line down
Enter – New line
Tab – Increase indent for selected lines
Shift+Tab – Decrease indent for selected lines
Ctrl+Up - Earlier command(s)
Ctrl+Down - Later command(s)
Ctrl+Enter - Run code

Single- or multiline input fields (in pop-up windows and property grid fields)

Tab – Next editable field
Shift+Tab – Previous editable field
Esc – Cancel editing
Enter – New line
Up – Line up
Down – Line down

Pop-up windows

Ctrl+Enter – Save data and close window
Esc – Cancel entry

Project panel context menu

F10 – Open data source in new panel

Data entity context menu

F12 – Open entity in new panel

4.3. Main menu

Be aware that through extension registration Main menu can be augmented with additional items. Refer to **Volume 3. Extensions** documentation volume for more details.

File

Project…

New project… (Ctrl+Shift+N) – saves current project state, closes all its data sources, creates a new project and asks for a project description file to keep state of a new project (the file should have *.15926* extension).

Open project…(Ctrl+Shift+O) – saves current project state, closes all its data sources and asks for a saved project description file to restore an older project (the file should have *.15926* extension). You can just drag Project description file and drop it on Project panel.

Save project - saves current project state.

Save project as… - saves current project state to a new project description file and continues a project under a new name.

Recent projects…- opens the list of recently closed projects. Hovering your mouse over items in the list you can see full file names with paths in the Status bar. The same list is available as clickable links on Start page with file names appearing as tips if mouse pointer is hovering over the name.

New data file… - creates new local data source and adds it to the current Project. User is asked for the mnemonic (non-unique) name of a data source. File location and file name are selected by user with the first *Save* command.

Add file(s)…

Choose data file(s)... - adds local data source with reference data (including template definitions) or with project data to the Project panel. File name(s) are used as a source name, and the default set of annotation properties (together with properties inherited from the project) is used for data source. From Project panel data source can be opened in data panel or panels.

The Editor is designed to work with RDF/XML files. Archives compressed by GZIP can be added without unpacking.

RDF files in Turtle format can be opened in the Editor as read-only (marked *r/o* in a Project panel and in data view root node). Use *Save as…* command to save them to RDF/XML for further work.

Combined data source can be created from multiple files by holding the *Shift* or *Ctrl* key in file selection dialogs. Combined data source is opened as read-only (marked *r/o* in a Project panel and in data view root node). Use *Save as…* command to combine data in a single file to start making changes.

You can select a data file or a group of files in MS Windows environment and drop it on Project panel.

Part 4 file … - Part 4 file is a representation of Part 4 tables in a legacy RDF format and can be downloaded from http://rds.posccaesar.org/2009/08/OWL/ISO-15926-4_2007.

PCA RDL file… - download and unpack ***PCA-RDL.owl.zip*** file from http://rds.posccaesar.org/downloads/PCA-RDL.owl.zip (the file contains copy of PCA RDL and is sometimes updated, use the newest version available). This data source is relatively big (more then 57 000 entities, almost 3 ml. triples), so you will see loading progress. You can continue work with the program while big data sources are loaded (though some tasks will be queued). You can also interrupt loading by *Ctrl+B* or by closing data source with cross button to the right of its name.

Proto and initial set templates file … - proto and initial template set in the ***p7tpl.owl*** file accompanying ISO 15926-8 is included with the Editor distribution in folder <*samples*>. The data source is added with default module name *p7tpl*. Any other template definition file can be added via this command if required.

Three menu commands above add standard data sources to the Project with default source names and specialized sets of annotation properties. Paths to the files can be registered via *File – Settings* menu command (*Paths* tab). If paths are not registered each menu command will ask for location of the file.

If paths to standard data sources are registered – they are added to the project as read-only (marked ***r/o***). If you want to change them – open the same files via *Add file(s)… - Choose data file(s)...* menu command or better use *Save as…* command to save a copy and edit it.

(Template file in Camelot format...) – starting from version 1.3 the Editor no longer supports import of template definitions in Camelot format (legacy RDF serialization formerly used in iRINGTools software). Probable you will never meet them anymore. If required – use old versions of the Editor to convert files or contact us for help.

Recent sources… - the list of data sources recently added to the Project for quick access after the closing. The list contains mnemonic (non-unique) data source names, not file names, if such names were assigned! Hovering your mouse over items in the list you can see full file names with paths in the Status bar. The same list is available as clickable links on Start page with file names appearing as tips if mouse pointer is hovering over the name.

Add SPARQL endpoint …

Choose endpoint - adds SPARQL endpoint data source to the Project with its URL as a source name and default set of annotation properties. If a SPARQL endpoint requires authorisation, name and password can be provided in the opening form.

Four useful SPARQL endpoints are directly available from this menu. They are added to the Project with default source name and specialized set of annotation properties.

PCA RDL - http://posccaesar.org/endpoint/sparql. Main endpoint for PCA reference data library supported and developed in the JORD project.

iRING Sandbox - http://www.iringsandbox.org/repositories/SandboxPt8/query. Sandbox for reference data developed in ISO 15926 Information Patterns project (http://iringug.org/wiki/index.php?title=ISO_15926_Information_Patterns_(IIP)).

***IIP Sandbox* (templates)** - http://posccaesar.org/sandbox/iip/sparql. Sandbox with base and specialized template sets developed and used in ISO 15926 Information Patterns project

TechInvestLab Sandbox - http://rdl.techinvestlab.ru:8891/sparql. Sandbox for reference data support of TechInvestLab.ru modeling methodologies.

Recent sources… - the list of endpoints recently added to the Editor for quick access after the closing. The list contains mnemonic (non-unique) endpoint names, not endpoint addresses, if such names were assigned! Hovering your mouse over items in the list you can see URLs in the Status bar. The same list is available as clickable links on Start page with file names appearing as tips if mouse pointer is hovering over the name.

From Project panel SPARQL endpoint can be opened in data panel or panels. An endpoint data source is always opened as read-only (marked r/o in a Project panel and in the data view root node.

Save (Ctrl+S) - saves data source selected in Project panel. If data source was not changed, is newly created, imported, or if data source is marked as *r/o* (read-only) – this command is disabled. *Use Save as…* command in these cases.

Save as… (Ctrl+Shift+S) - saves data source selected in Project panel to a new RDF/XML file. If multiple files were added as a single data source, they are saved to a single file. If data source is a SPARQL endpoint, all data items downloaded from it at the moment (templates or reference and project data) are saved to a file. It is recommended to save reference and project data files with *.rdf* extension, template definition files – with *.owl* extension. File type ***Compressed GZIP file (*.gz)*** can be selected to save disk space, you can enter name with *.rdf.gz* or *.owl.gz* extension in this case. If a file with selected name and extension already exists overwriting confirmation will be requested.

Save all (Ctrl+Alt+S) – saves all unsaved data sources in Project panel, except new or imported data sources for which no file names are assigned yet, and saves current project.

Save snapshot...- saves as a new file all data items downloaded from a SPARQL endpoint (templates or reference and project data) and adds it as a new data source to the Project panel. For local file data sources the command works like *Save as…* command. If the file with such a name already exists overwriting confirmation will be requested.

For commands above make sure that required data source is selected in Project window when you access the File menu; otherwise you'll probable see menu commands disabled or find your commands take effect on unexpected data source.

Stop task (Ctrl+B) – interrupts time consuming load, save, import or search task (description of task currently executed by the Editor is displayed in the Status bar). Not all tasks can be interrupted though.

Reimport (Ctrl+Shift+W) – reimports the data for data source created from imported template definitions or through *Import* menu.

Reload modules (Ctrl+Shift+R) – recompiles and reloads extension modules.

Reload patterns (Ctrl+Shift+W) – reloads pattern definitions from pattern files used by the Editor, allowing to add, edit and test patterns without closing the Editor.

Settings – access the form for registration of extensions and localization toggle. *Paths* tab… also allows path registration for external Python libraries (for use in extensions and Python console) and for three standard data sources opened from *File – Add file(s)…* menu. Settings are saved between Editor launches as part of the Editor's configuration (dot15926.cfg file in <installation_folder>).

Exit (Alt+X) - exits application saving current project, Editor window location and general panel layout. Notice that no information about data views or open panels is saved! The notification will appear if you have unsaved files (but not for unsaved endpoint data!).

Edit

Be sure that required data panel is active when you access the *Edit* menu; otherwise you'll probable see menu commands disabled or find your commands take effect in unexpected data panel.

Undo (Ctrl+Z) and ***Redo (Ctrl+Y)*** commands provide a safe editing environment. Undo and redo data is saved for changes in data and properties of each data source.

Copy (Ctrl+C), ***Copy text (Ctrl+Shift+C)***, ***Cut (Ctrl+X)***, ***Paste (Ctrl+V)*** and ***Paste as triples (Ctrl+Shift+V)*** commands allow to move reference and project data entities and their properties, whole template definitions, or text from data panels between appropriate data sources and external applications. Use of a particular command is context-dependant and governed by specific rules documented in corresponding section of this documentation.

Insert new item (Ins) – adds new properties to entities and new roles to templates (notice that this command is not used to create any new reference and project data entities!).

Add template (Shift+Ins) – adds new template (base or specialized) to a data source.

Delete item *(Del)* – deletes data items (data entities, properties, templates and roles, search results).

Reload item (F5) – reloads all properties (roles, annotations, relationships) of data entity in focus and reidentifies all patterns it is part to; if focus is on a grouping node or on a single data entity obtained through local search or SPARQL endpoint query - repeats the query (useful in case of local changes made or in case of slow or incomplete response from remote server). Beware, the command refreshes representation of items existing in Editor's memory or on a remote endpoint and will not reload the data from the local file(s)!

Put new UUID onto the clipboard (Alt+U) – generates and puts onto the clipboard an alpha-numerical string by concatenating prefix (with default value "id") to the UUID compliant to RFC 4122 / ITU-T X.667 / ISO/IEC 9834-8. This command can be used to populate property fields which require unique identification strings. Generated string will float up in the lower right corner of the Editor window and stored in the Console history.

Search

Search endpoints for URI (F4) – searches all SPARQL endpoint data sources available in Project for URI of an item in focus (URI of a data entity, URI of an object property occupier, or URI of template role restriction). If search is successful - search result is added as a new data view for corresponding endpoint data source in Project. If no results are found – the message will float up in the lower right corner of the Editor window and stored in the Console history.

Search PCA endpoint for WIP equivalent (F7) – searches PCA SPARQL endpoint http://posccaesar.org/endpoint/sparql for data entity equivalent to legacy RDS/WIP entity in focus (URI in the namespace *http://rdl.rdlfacade.org/data#*). Search results are added as new data view for PCA endpoint data source in Project panel. If PCA endpoint is absent from Project panel – it will be added automatically. If no results are found – the message will float up in the lower right corner of the Editor window and stored in the Console history.

Open URI in web browser (F6) – launches your system's web browser and tries to dereference an URI of an entity in focus (or URI of an occupier of an object property in focus) via your web connection. The result depends on whether URI is dereferenceable or not.

Open property URI in web browser – launches your system's web browser and tries to dereference an URI of a property in focus via your web connection. The result depends on whether URI is dereferenceable or not.

All entities from data source – searches current data source (local file(s) or an endpoint) for all reference and project data entities (all entities used as subjects in RDF triples). Returns both entities with URI (named or not named) and blank nodes.

All templates from data source – searches current data source (local file(s) or an endpoint) for template definitions. Template definitions are encoded in a specific way by complex subgraphs of RDF graph. This command identifies all template definition subgraphs in a local file(s) or on an endpoint, and represents them in a data tree.

Search for suspicious entities – searches for all entities which do not pass built-in data verification tests. For descriptions of currently implemented test refer to Data verification section. This command execution can take a lot of time for big data sources!

View

Open in new panel (F12) – in the Editor you can create many views for a single data source. This command creates the new view for data item in focus (which contains this item as the only node below the root node) and opens it in a new panel. You can start navigation from this node or add other nodes to this view through search/filter field, Console queries or editing.

Properties nodes, Relationships nodes– toggles corresponding grouping nodes (*Properties* and *Relationships)* in a data tree of any local data source or endpoint. Useful for compact viewing of data trees. When output of grouping nodes is turned on, they will not be identified automatically for entities unfolded earlier, use *Reload item* menu command on such items or hit *F5.*

Patterns (Ctrl+P) – toggles pattern identification and output of corresponding grouping nodes in a data tree of a local data source. Turning patterns off can be useful for compact viewing of data trees and can save processing time if complex patterns are defined and selected for visualization in project properties. When pattern identification is turned on, patterns will not be identified automatically for entities rendered earlier, use *Reload item* menu command on such items or hit *F5.*

Simplified entity view (Ctrl+P) – toggles ungrouped pattern view in a data tree of a local data source. In a simplified view all grouping nodes in a data tree (Properties, Relationships, Patterns nodes and grouping nodes for individual patterns) are removed and all identified patterns are visualized directly under the entity node. Only patterns selected for visualization in project properties are identified. If any of the toggles for Properties, Relationships or Patterns is turned on – simplified view is turned off. Simplified view for an endpoint shows only entity labels without any additional data.

Python console (Ctrl+`) (***Ctrl+backquote*** at upper left corner of standard keyboard) - toggles Python Console panel.

Project panel (Ctrl+T) - toggles Project panel.

Properties panel (Ctrl+G) – toggles Properties panel.

Start page – toggles Start Page.

Toolbar – toggles Toolbar.

Font size – Editor font size can be changed in this submenu or by *Ctrl*+= and *Ctrl*+- keyboard shortcuts, *Ctrl+0* (*Ctrl-Zero*) restores default font size. Font size adjustments are convenient for public presentations.

Data types

Data types menu commands allow access to built-in data type sets used to organize ISO 15926 data. Each data set is added as a data source to the Project panel and can be opened in data panel or panels. Data type sets will be opened as read-only and without possibility to save them to a new file.

XML Schema – contains literal property type set from http://www.w3.org/2001/XMLSchema# namespace.

Part 2 – contains upper ontology of ISO 15926-2 lifecycle integration schema with subtype/supertype structure, properties/roles, definitions, dependencies, restrictions and three sets of URI's. For each data type URIs in three namespaces are known to the Editor:

PCA RDL
http://rds.posccaesar.org/2008/02/OWL/ISO-15926-2_2003# ;

RDS/WIP
http://dm.rdlfacade.org/data# ;

ISO 15926-8
http://standards.iso.org/iso/ts/15926/-8/ed-1/tech/reference-data/data-model# .

Import

This is an extension menu listing built-in extensions from TechInvestLab.ru. Particular extensions may be turned off through *File-Settings* command. Order of menu items may vary. Refer to **Volume 3. Extensions** section for details.

Catalog import is an open-source built-in extension for import of reference data from data model of an engineering catalog application developed by 3V Services company.

> ***Setup catalog dependencies…*** - opens dialog for catalog data import settings (settings are saved between Editor launches in the ***dot15926.cfg*** configuration file in <*installation_folder*>).
>
> ***Import catalog data from JSON file…*** - select catalog JSON file for import.

Template import is an open-source built-in extension for import of new template definitions from spreadsheet format used by iRINGTools software, including generation of URIs for templates and roles .

> ***Import templates from .xlsx table…*** - select spreadsheet in .xlsx format for import.

IIP template import is an open-source built-in extension for import of existing template definitions (together with URIs) from spreadsheet used to populate iRING Tools software.

> ***Import IIP templates from .xlsx table…*** - select spreadsheet in .xlsx format for import.

Spreadsheet mapper is an open-source built-in extension for mapping spreadsheets to patterns and for import of reference and project data from spreadsheets. For more details refer to **Volume 4. Patterns and Mapping**.

> **Build patterns from Excel** – opens form for the selection of spreadsheet and pattern, mapping definition and import.

TabLan import is an open-source built-in extension for import of reference and project data from TabLan.15926 table language.

> ***Setup TabLan dependencies…*** - opens dialog of TabLan model import settings (settings are saved between Editor launches in the ***dot15926.cfg*** configuration file in <*installation_folder*>).
>
> ***Import TabLan reference data from .xlsx table …*** - select TabLan .xlsx file for import.

Help

About – brings up a window with basic version information.

Check for new version – the Editor tries to access http://techinvestlab.ru and check whether newer version is available. If newer version is found – opening of download page in internet browser is suggested.

Documentation – looks for a documentation folder defined at installation and opens it if found. Tries to access documentation page at http://techinvestlab.ru if local search fails.

4.4. Toolbar

Buttons in block 1 execute *File* menu commands: ***Save***, ***Save all***

Buttons in block 2 execute *Edit* menu commands: ***Undo, Redo, Copy, Cut, Paste, Insert new item, Delete item, Reload item***.

Buttons in block 3 execute *Search* menu commands: ***Search endpoints for URI, Search PCA endpoint for WIP equivalent, Open URI in web browser, Open property URI in web browser, All templates from data source, All templates from data source***.

Buttons in block 4 execute *View* menu commands: ***Open in new panel, Python console***.

4.5. Project panel

Project is a main data organization concept of .15926 Editor. Data sources (local files and endpoints) are added to the Project and linked together – whenever URI of some object is met in one data source, it is automatically identified in other data sources (for local files) or can be searched with one key strike (for endpoints).

The project has a set of properties - module names, object and annotation properties, inherited by all data sources in the project (see below for details). Data sources in the project have their own properties also. Project state (project properties, list of data sources and their locations and data source properties) is saved to the project description file (local file with ***.15926*** extension). Standard data sources with registered paths added via *Add file(s)…* menu command are saved to the project as references to the standard files. If a project file is opened on another installation of the Editor – such project will attempt to open standard file(s) of the current installation!

Open data views and content of data panels are not saved with the project!

The name of the project description file is used as a project name. Project name is a top node in the data source tree rendered in the Project panel. Project name also forms part of the application name in MS Windows environment and is a part of application window title.

If composition of project properties or data sources are not saved – the project is marked with * in Project panel and in application name. You can save a project by *File – Project… – Save project* menu command or use *File - Save all* menu command will save the project together with all unsaved data sources.

Project autosave is enabled in the Editor. If the program stops working – it will ask you to restore the project at the restart. Only project data is restored, unsaved changes to data sources will be lost.

To open previously saved project you can just drag a Project description file in MS Windows environment and drop it on the Project panel.

Current .15926 Project is saved as part of the Editor's configuration every time the Editor is closed, and restored at a next launch. Configuration is saved to the ***dot15926.cfg*** file located in the main folder with Editor executable (<*installation_folder*>).

Project panel is a home for the project. Data sources can be added to the Project panel via *File* and *Data types* menus, or directly dropped to it from MS Windows environment (a local file or a group of local files). Project panel lists all data sources of the Project and all opened data views. Project panel helps in navigation between data sources and data views.

Project panel tree has a top node  icon representing current project. Data sources are represented as nodes under the project node.

Each data source node has a corresponding data view (it can be rendered in a data panel or not rendered at any given moment). Every time a new data source is added to the project it appears as a node in the Project panel. Mnemonic (non-unique) names of data sources are used in Project panel to identify data sources. Hovering mouse over the data source it is possible to see as a tip the full path(s) for local file(s) or URL for endpoint.

The kind of the data source is indicated by an icon with a letters:

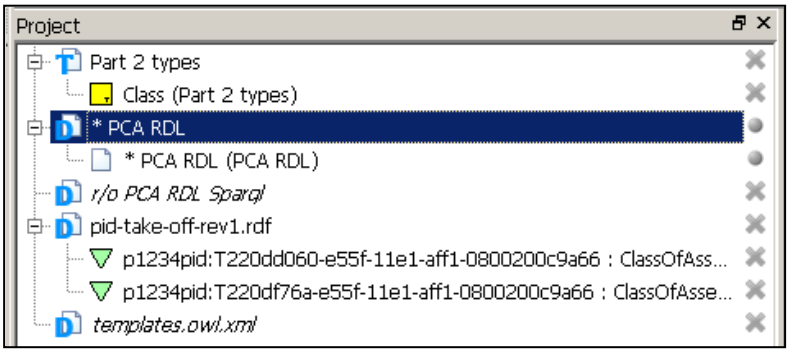
 - D, reference and project data;

 - T, built-in data types.

Data sources with unsaved data changes are marked with ∗. Data sources with unsaved changes to data source properties are marked with *italics*.

Nodes below each data source node are separate data views opened from this data source. They can be data views of the same data source, diff views built for comparison with some other data source, or views of other linked data source. Multiple initial views of the same source can be opened. Non-initial views opened from particular data items are marked with item's icon and named after it.

Cross *Close* buttons to the right of Project panel nodes will close data views or close and remove data sources. It is possible to interrupt big data source loading or *search* box query by closing the data source. It is often impossible to interrupt a Console query by closing the data source. It is always better to try *File-Stop task (Ctrl+B)* menu command first.



The form of *Close* button is changed from cross to circle for unsaved data sources.

On the screenshot there are five data sources added to the project:

Part 2 *types*, *PCA RDL* as local file, *PCA RDL Sparql* as remote endpoint, project data file *pid-take-off-rev1.rdf* and *templates.owl.xml*. Remote endpoint is marked as read-only. Additional initial data view is opened from *PCA RDL* and separate views on particular elements are opened from other data sources. *PCA RDL* is marked with ∗ as its data is changed and changes are not saved to the data source. *PCA RDL Sparql* and *templates.owl* have names in *italics* because their properties are changed and changes are not saved to the project description file.

Closing of any views do not affect data source (therefore no confirmation will be asked). If you close and remove a data source node from the project all views opened from it will be closed and data source will be removed from the Editor environment. If this data source is a local file and there are unsaved changes to its data - you will be prompted to save it. If this data source is an endpoint – no confirmation will be asked. Be aware that URIs used in some other data sources may not be visualized properly if you remove data source from the project.

Navigation in Project panel is possible with arrow keys.

There are multiple ways to open a view from a Project panel node or data tree node:

To open data view in a tab of an active data panel:

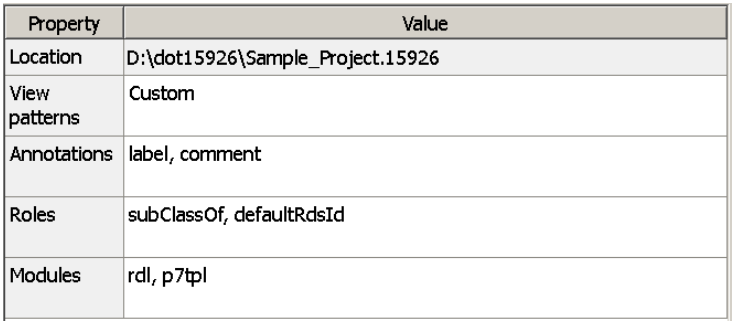
- double click on a node
- click the right mouse button on a node and select *Open*;

To open data view in new panel:

- click the middle mouse button on any node;
- click right mouse button on any node and select *Open in new panel command*;
- press *F10* while cursor focus is on a data source.
- press *F12* while cursor focus is on a node in data tree.

If a view is already opened in some panel, an attempt to open it by double click or *Open* command will make this panel active and its title bar will flash to attract your attention.

4.6. Project properties



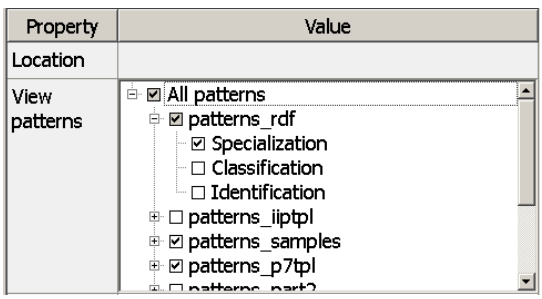
Property	Value
Location	D:\dot15926\Sample_Project.15926
View patterns	Custom
Annotations	label, comment
Roles	subClassOf, defaultRdsId
Modules	rdl, p7tpl

You can access project properties in a Properties panel if you click project node in a Project panel.

You can start editing editable properties by double clicking the field with property values. You can save changed properties by clicking anywhere outside of the field. Pressing *Esc* abandons unsaved changes. You can use *Undo* and *Redo* commands after you've saved changes to project properties.

Properties of the project are:

Location – full path to the project description file (local file with ***.15926*** extension). The field is not editable.



View patterns – pattern identification in the data source can take significant time. Using this field you can determine which patterns will be identified and visualized in data panels for each entity opened (patterns are visualized for local data sources only).

Patterns libraries are located in Python files with ***.py*** extension, placed in the <*installation_folder*>*/patterns* folder. The Editor is distributed with initial pattern library ***patterns_samples.py***,

Initial pattern set defined in this library depends on three template data sources (see below for module names help):

- Part 8 initial set in a module named ***p7tpl*** (the file ***p7tpl.owl*** accompanying ISO 15926-8 is included with the distribution in folder <*samples*>);
- PCA MMT SIG set in a module named ***projtpl*** (the work-in-progress version of PCA MMT SIG set is included with the distribution as ***templates.owl*** file in folder <*samples*>*pid*);
- IIP template set in a module named ***iiptpl*** available as **IIT Sandbox (templates)** from *Files* menu.

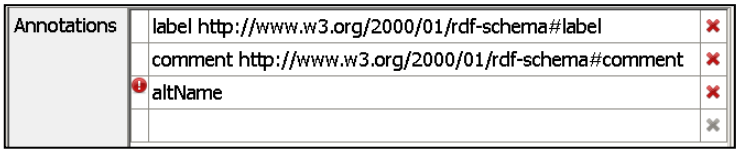
IIP template set in a module named ***iiptpl*** available as **IIT Sandbox (templates)** from *Files* menu.You can download other pattern libraries from our site as they are released or develop them yourself. Refer to **Volume 4. Patterns and Mapping** for more details.

Starting editing this field you will see pattern libraries organized in a tree. You can use checkboxes to select pattern for visualization in data panel. Unfolding library nodes you can see individual patterns defined in each library and fine-tune their visualization (note that different options for a single pattern can be defined in different libraries). You can use *All patterns* checkbox to quickly select all or none patterns.

Property values shown for the saved *View patterns* property are just *All*, *Custom* or *None*. You have to start editing to find exact settings for *Custom* value.

Independent of the selection made in this field you can search in the console for any pattern from any library loaded in the Editor.

Annotations – annotation properties (short names and URIs) for use in all data sources of the project. Default set of project annotations includes only ***rdfs:label*** and ***rdfs:comment*** with ***label*** and ***comment*** short names respectively.



If you know URIs of other annotations you would like to use for search and editing in all data sources of the project – you can add them to project *Annotations*. Start editing; put new annotations on blank lines as space-separated pairs **name URI**. If format is wrong – the Editor will indicate an error and will not allow to save the data. The Editor will not check whether URI is well formed or not!

You can use *Cut*, *Copy* and *Paste* commands to move *Annotations* between project and data source properties. Click on the respective line and use a command or keyboard shortcut. Delete unnecessary annotations with red cross *Delete* button at the right of the line.

Property values shown for the saved *Annotations* property are short names of the annotations. The short names can be redefined for particular files or endpoints in the project.

Roles – custom object properties (short names and URIs) for use in all data sources of the project. Custom object properties are not defined in Part 2 data model and can be used to record relationships in a non-reified way. Default set of project roles includes only ***rdfs:subClassOf*** with short name ***subClassOf****,* it is sometimes used to record specialization relationship.

If you know URIs of other custom object properties and would like to use them for search and editing in all data sources of the project – you can add them to project *Roles*. Start editing; put new roles on blank lines as space-separated pairs **name URI**. If format is wrong – the Editor will

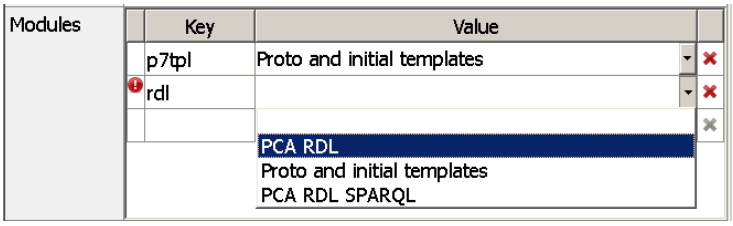
indicate the error and will not allow to save the data. The Editor will not check whether URI is well formed or not!

You can use *Cut*, *Copy* and *Paste* commands to move *Roles* between project and data source properties. Click on the respective line and use a command or keyboard shortcut. Delete unnecessary roles with red cross *Delete* button at the right of the line.

Property values shown for the saved *Roles* property are short names of the roles. The short names can be redefined for particular project files or endpoints,

Modules – for any data source a unique module name can be defined in the Editor environment for referencing in API function calls of various .15926 Platform components. Module name should be an alphanumerical identifier starting with a letter and containing no spaces (underscores are allowed). For information on module name usage refer to **Volumes 2** and **4** of this documentation.

Module name is predefined for standard Part 8 template data source (*p7tpl*) and can be defined manually for other data sources of the project.

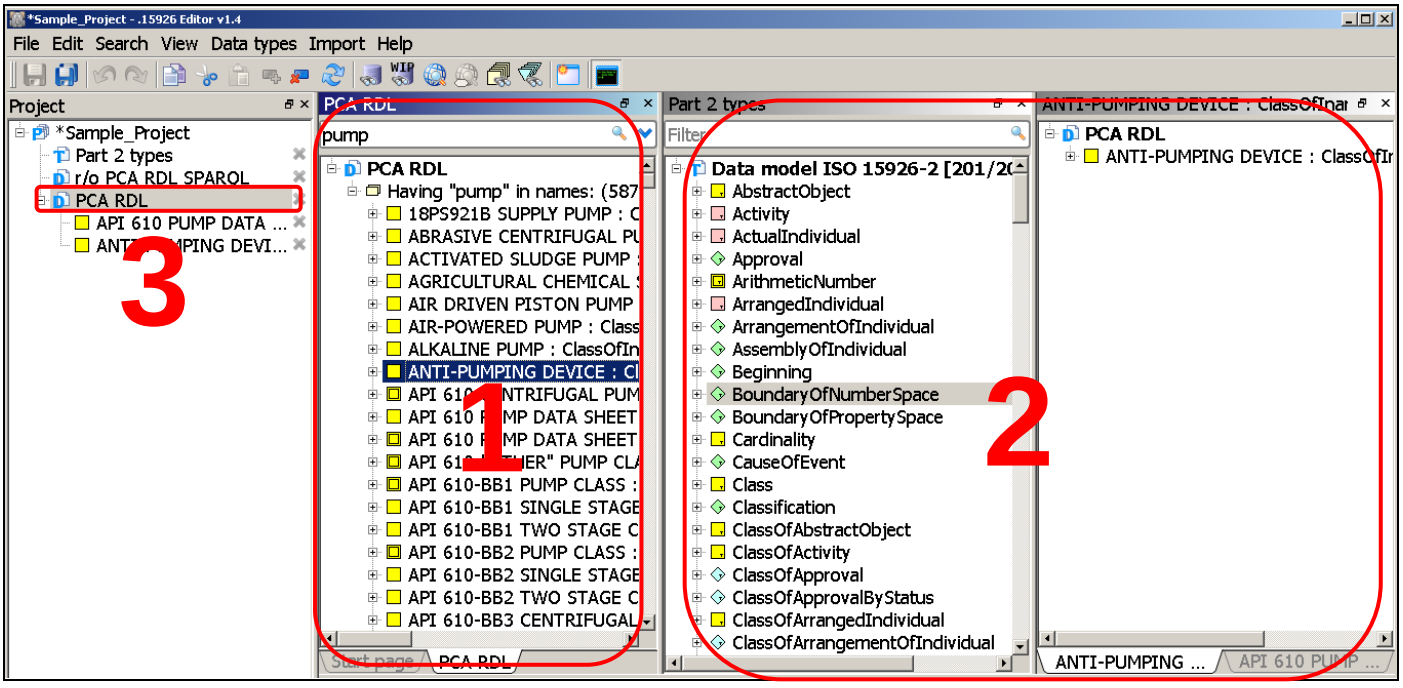


Starting editing you will see the list of all named modules. To add a new module name enter a new name in a key field (see above for requirements). The Editor will mark the entry as incomplete. Click on the drop-down menu and choose a data source. The Editor will mark as errors all non-unique module names, and will check whether the name is formed correctly! If any mistakes are found – The Editor will not allow you to save properties.

Delete unnecessary modules with red cross *Delete* button at the right of the line.

Property values shown for the saved *Modules* property are module names of the project.

4.7. Data panels



Data panels are used for data source viewing and editing. Data panels can be stacked on top of each other for tabbed access

Data panel can be active or inactive. You can see active (1) and inactive (2) data panels on screenshot above. Pay attention that corresponding data source in Project panel (3) is selected automatically when you make panel active.

An active panel has a blue title bar while inactive panels have a grey one. Clicking on a tab, on a title bar or anywhere inside a visible panel makes it active. At program launch no data panels are created, only Start page is visible. Data panels are created if opening of a data view is requested.

Data panels can be undocked from the main window by clicking the *Undock* button on the title bar of the panel or by dragging the title bar outside of the main window with mouse. Panels are docked again by dragging the title bar inside the main window. To rearrange or stack data panels together drag them by the title bar across the main window. Allowed places for panels will be shown during dragging; dropping on the existing panel of an appropriate type will create a tab. To close data panel click the cross button on the title bar of the panel. Closing the panel do not affect data source and data view rendered in it, they remain listed in the Project panel and can be reopened from it.

4.8. Properties panel

Properties panel contains property grid where you can see all properties of an element in focus (element can be a project, a data source, a grouping node, a template definition, a data entity, a property, a role, etc.).

You can select text in any field of the grid and copy its content with *Ctrl+C* keyboard shortcut.

If a field in a property grid is white – you can edit it directly in the grid. Click once on a text field or use double-click to start editing properties with list values.

If you are editing a multiline text field – *Enter* will insert a new line. *Tab (Shift+Tab)* will save your changes and move to the next (previous) value field of a property grid. Click on any other field will also save changes. *Esc* will abandon all unsaved changes made to the field. You can use *Undo* and *Redo* commands after you've saved changes to properties.

If a property field is grey – you can't edit it in the grid, but sometimes you can change it in the tree by drag-n-drop process.

Content of a property grid for each kind of an element is described in the corresponding section of this documentation.

4.9. Console

4.9.1. REPL Environment

Python console is one of the most powerful features of .15926 Platform and can be built into any application developed on it. Scripting in Python brings all power of general purpose programming language to data modeller or data miner. The Console is a fully-functional Python REPL environment which allows user to run simple or complex scripts in Python, and advantages of Turing-complete language are hard to overestimate. The console scripts can access API functions of various .15926 Platform components to read, explore and change reference and project data.

Python console is opened through menu *View*, with *Ctrl+`* (*Ctrl+backquote* at upper left corner of standard keyboard) keyboard shortcut or with a toolbar button.

Console panel has history area (top part) and input area. Type a Python expression in input area and hit *Ctrl+Enter* to run it. Try:

print "Hello world"

or

2+22**

The code executed in the Console has to comply with Python syntax.

You can use **help()** command to access Python help utility in the console. A prompt **help>** will appear in history area. Enter commands under this prompt to get lists of modules, their data structures, classes and methods (degree of component documentation can vary). Enter **quit** to leave help and return to the REPL environment.

4.9.2. Console interface

Console input is saved to ***input.log*** file, output is saved to ***dot15926.log***, both files are located in the main folder with Editor executable (<*installation_folder*>).

History area is also showing program's diagnostics, storing error messages and notifications.

The history area allows *Ctrl+C* copy command. Please include console output when reporting problems encountered during software use. If the program closes unexpectedly, error messages (if any) can be copied from the ***dot15926.log*** file located in the main folder with the Editor executable (last messages are at the end of the file).

Console allows input and execution of a complex Python language script. In input area *Enter* will create a new line, up and down arrows navigate in the input area. *Run* and input history buttons are at the bottom of input area, or use *Ctrl+Up* and *Ctrl+Down* to access history, *Ctrl+Enter* to run the code.

Run file button brings file selection dialog and executes Python code from selected file. Code executed from the file is not recorded in input history.

Designing complex Python scripts it is ultimately necessary to manage indentations. You can click at the start of the line or select several lines of text and increase their indent by pressing *Tab* or decrease the indent with *Shift+Tab.*

4.9.3. Console scripting

If an API function of some .15926 Platform component is called in the console, it will be applied:

- Either to the data source open in the current active panel,
- Or to the data source identified via ***with context(…):*** statement.

By default API function called from the console is applied to the data source open in the current active panel (except functions called as methods of an identified module). If you are going to search and/or modify several data sources simultaneously and interrelatedly, each API function call (or several calls) should be wrapped in a

with context("module_name"):

statement. Wrappings can be nested. Any call outside of inner wrapping is again applied to the data source defined by outside wrapping, if any, or to the data source in the current active panel. Please refer to the Project properties section for details of module name assignment. See usage examples below.

To learn about functions available in Python console refer to the **Volume 2. APIs of Scanner and Builder** of the documentation, or read about libraries and modules in **Volume 3. Extensions**. For specific restrictions on data sources available to various Platform components also refer to the corresponding API specifications.

If ***"NameError: name '…' is not defined"*** error message is displayed on the console history panel – check whether indeed proper data source is open in your active panel or in a module you are addressing through ***with context(…):*** statement.

Please remember that changes to the data source effected through console function calls could not be undone!

Some Python libraries are included in the Editor at compilation. If you want to use external Python libraries in your console code, the Editor may have troubles locating them. You can either copy all required libraries to the *<installation_folder>*, or add paths to them in *Python search paths:* field (on the *Paths* tab in the *Settings* form) as comma separated list. *Settings* form is available through *File-Settings* menu command.

4.9.4. Example scripts

a. Assume you have a need to send an email to a colleague with designations of all reference data entities in PCA RDL which are instances of Scale type and have "celsius" substring in their labels.

Open PCA RDL file (or its copy – for safety) via *Add file(s)… - Choose data file(s)...* menu command so that you'll be able to change it.

Paste the following script into Console (please honour leading spaces as they are principal feature of Python syntax, we are sorry but **some .pdf readers do not allow you to copy them properly**).

```
outfile = file('scales_celsius.txt', 'w')
scales=find(type=part2.Scale, label=icontains('celsius'))
for entity in scales:
   designations=find(id=entity, hasDesignation=out)
   for property in designations:
     outfile.write(entity+' : '+property+'\n')
outfile.close()
```

Press *Run* button (or hit *Ctrl+Enter*). Look for ***scales_celsius.txt*** file in the main folder with Editor executable.

b. Let's analyze patterns of data in RDL to identify all possible instances of the template ClassOfCauseOfBeginningOfClassOfIndividual and form a list of possible signatures in a separate file. Run the following script:

```
outfile = file('COCOBOCOI_sign.txt', 'w')
begun=find(type=part2.ClassOfCauseOfBeginningOfClassOfIndividual,
hasClassOfBegun=out)
for elem1 in begun:
   causer=find(hasClassOfBegun=elem1, hasClassOfCauser=out)
   for elem2 in causer:
     outfile.write('ClassOfCauseOfBeginningOfClassOfIndividual('+elem1+','+elem2+')\n')
outfile.close()
```

And look for ***COCOBOCOI_sign.txt*** file in the main folder with Editor executable.

c. Now we'll use two APIs available from console – Scanner and Builder APIs. Make sure that Proto and initial templates were added to the Project panel through *File - Add file(s)… - Proto and initial set templates file …* menu command (to get a proper module name). Open PCA RDL file in an active panel and run the script:

```
beguns=find(type=part2.ClassOfCauseOfBeginningOfClassOfIndividual,
hasClassOfBegun=out)
for element1 in beguns:
   causers=find(hasClassOfBegun=element1, hasClassOfCauser=out)
   for element2 in causers:
     p7tpl.ClassOfCauseOfBeginningOfClassOfIndividual(hasClassOfBegun = element1,
      hasClassOfCauser = element2)
```

Instead of writing template signatures to a text file it really adds RDF representations of template instances to PCA RDL data, as you can check with the query:

```
show(type=p7tpl.ClassOfCauseOfBeginningOfClassOfIndividual)
```

Don't forget to delete the new entries with a script (there are no other template instances in PCA RDL):

```
added_templates = find(type=p7tpl.ClassOfCauseOfBeginningOfClassOfIndividual)
builder.delete(added_templates)
```

(or just close file without saving).

d. Let us do the same change in a clean way – create new template instances in a new data source. Assign module name ***pca*** to PCA RDL data source, add new data source to the Project

panel via *File - New data file...* menu command (any data source name will do) and assign it the ***new_data*** module name.

Run the following script (now it does not matter which panel is active or whether there are open data views at all):

```
with context("pca"):
  beguns=find(type=part2.ClassOfCauseOfBeginningOfClassOfIndividual, hasClassOfBegun=out)
  for element1 in beguns:
    causers=find(hasClassOfBegun=element1, hasClassOfCauser=out)
    for element2 in causers:
      with context("new_data"):
        p7tpl.ClassOfCauseOfBeginningOfClassOfIndividual(hasClassOfBegun = element1,
        hasClassOfCauser = element2)
```

You can check now that new template instances were added to your new file, not to PCA RDL data source.

Another example script looking for patterns of ClassOfIndirectProperty template:

```
with context("pca"):
  spaces= find(type=part2.Classification, hasClassifier=out,
  hasClassified=find(type=part2.PropertyQuantification, hasInput=out,
  hasResult=find(type=part2.RealNumber)))
  for element1 in spaces:
    posessors=find(type=part2.ClassOfIndirectProperty, hasClassOfPossessor=out,
    hasPropertySpace=element1)
    for element2 in posessors:
      with context("new_data"):
        p7tpl.ClassOfIndirectProperty(hasClassOfPossessor=element2,
        hasPropertySpace=element1)
```

These scripts can be used in data transformation extension provided as an extension example and described in **Volume 3. Extensions**.

4.10. Status bar

Description of task currently executed by the Editor is displayed in the Status bar. Time consuming load, save, import or search task can be interrupted through *File - Stop task (Ctrl+B)* menu command. Not all tasks can be interrupted though.

Status bar also shows full filenames or URLs for items from *Recent…* lists in *File* menu (the lists themselves contain mnemonic (non-unique) data source names).

4.11. Data view

Each data panel with an open data view has a *filter/search* box on top and a data tree below it.

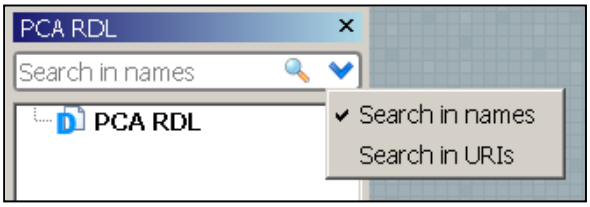
4.11.1. Filter/Search box

Each data panel with an open data view has on top a *filter* box or a *search* box.

Filter box appears for built-in data type sets and for diff views. For built-in data type sets filtering is performed on the fly as each new character is added to the *filter* box. For diff views filtering requires *Enter* key or mouse click on a magnifying glass icon.

For data sources with *filter* box an element counter is located after the data source root node. Element counter shows number of items displayed with current filter and total number of items in the filtered data source view.

Reference and project data sources have a *search* box. In initial view for these data sources the data tree is represented by data source root node only. After opening of an empty initial view of a data set, try using the *search* box.



Search box has two operating modes:

- Search in names
- Search in URIs

The modes are changed by a click on a blue icon for drop down menu at the right end of the search field.

There is no wildcard support in the released version; all searches look for a substring (case-insensitive). Please be careful with *Empty search* (strike *Enter* with cursor in the empty box) for really large remote data sources (endpoints)! Search box queries for local data sources can be interrupted by *File-Stop task (Ctrl+B)* menu command.

Use menu command *Search - All templates from data source* or corresponding Toolbar button to look for all template definitions which may be present in a local data source or on an endpoint. This command looks for complex RDF subgraphs used to represent templates and renders them in a data tree.

Search in names

For SPARQL endpoints name search in the *search* box is looking only for *rdfs:label* property.

For local data sources (files) name search in the *search* box is looking for names of reference and project data entities. Names of entities can be represented by different literal and annotation properties in different data sources. Properties are searched in the following order:

1. *rdfs:label*
2. *pca:hasDesignation*
3. *dm:hasContent*
4. *meta:annUniqueName*

If the property from this list is present for an entity and doesn't match search condition, subsequent properties for this entity are not checked. If a property is absent, the search checks for the next property. The name search returns an entity as soon as some property value matches the search condition.

Empty search (strike *Enter* with cursor in the empty box) in name search mode looks for all named entities in the data source.

If naming of entities in a reference and project data source is done via some other annotation property, through identification relationships or with identification template instances, name search

in a *search* box will bring nothing. Use **show(type=part2.any.Thing)** or **show(type=icontains(''))** query in the Console to find all typed entities in local file data source, or use an URI search.

Search in URIs

URI search looks for a search string in URIs of all reference and project data entities (all entities used as subjects in RDF triples of a data source). It can also return blank nodes of the original graph as it is also looking through blank node IDs.

Empty search (strike *Enter* with cursor in the empty box) in URI search mode is equivalent to *Search – All entities from data source* menu command or to **show(id=icontains(''))** query in the Console.

4.11.2. Data tree

1		Data source or grouping node
2		**Thing, AbstractObject** and all its subtypes, except: **MultidimensionalObject,** relational data types (**ClassOfClassOfRelationship**, **ClassOfRelationship**, **Relationship**), **ClassOfClass** and their subtypes
3		**ClassOfClass** and its subtypes, except relational data types
4		**MultidimensionalObject, PossibleIndividual** and its subtypes
5		**ClassOfClassOfRelationship** and its subtypes
6		**ClassOfRelationship** and its subtypes
7		**Relationship** and its subtypes
8		Instance of types listed in row 2 above
9		Instance of types listed in row 3 above
10		Instance of types listed in row 4 above
11		Instance of types listed in row 5 above
12		Instance of types listed in row 6 above
13		Instance of types listed in row 7 above
14		**Template** type
15		Template definition (subclass of **BaseTemplateStatement)**
16		Specialized template definition (subclass of **RDLTemplateStatement**)
17		Template instance
18		Recognized non-ISO 15926 typed entity (XSD Schema, some OWL classes)
19		Registered annotation or literal property (standard or custom)
20		Standard object property
21		Unregistered annotation property
22		Unrecognized entity (with absent, improbable or unknown type)

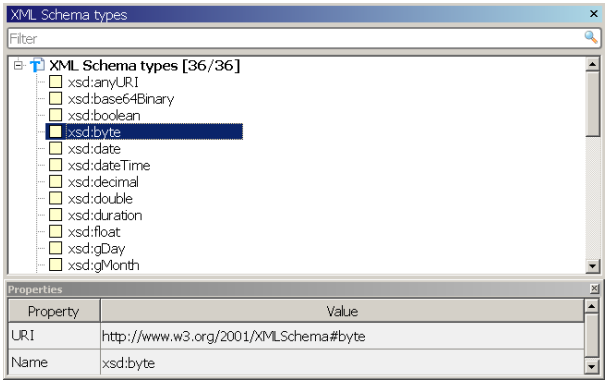
Nodes in data trees are marked with icons. See the table at the left for the list of icons you can find in the tree and their meanings.

Many nodes contain plus sign before them which can be unfolded to explore the data tree further. Sometimes the plus sign before an entity will not allow unfolding – meaning that an entity from other data source is referred here (or that all additional node groups are turned off via *View* menu). If an additional data source is present in the project, *F12* will open that entity in a separate data view where it can be explored. Or you can search for an entity at available endpoints with *F4.*

Nodes representing groups of entities contain counter of subnodes in the group. If counter is greater then zero – the group can be unfolded. Working with an endpoint you may notice that counter value changes for some time as Editor receives additional replies from an endpoint.

If a group of entities in a reference and project data source tree contains more then 1000 entities they are displayed in portions to speed up an output. Please click ***…more*** node at the end of the tree to see the next thousand.

4.11.3. Data tree overview

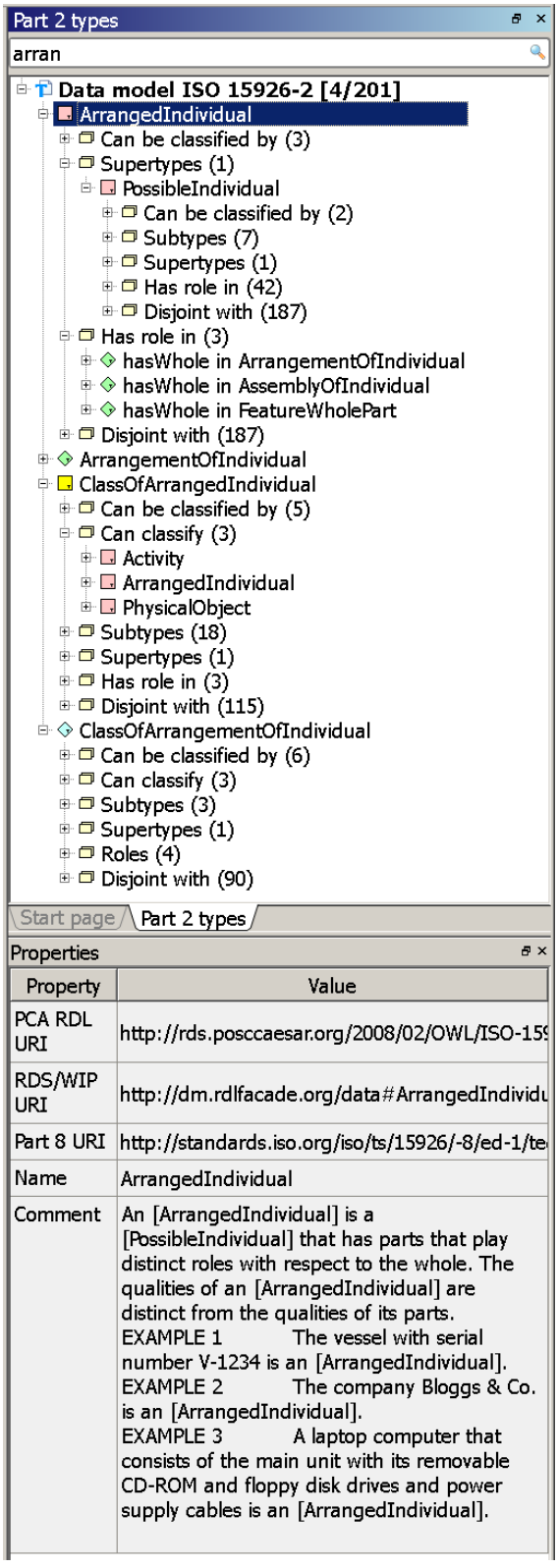


We have several types of data sources and each can contain data trees with different nodes. Tree node kinds, information available at their unfolding and their properties are described below.

XML Schema type

XML Schema type data view contains data types for representation of strings, numbers, dates and the like. Tree node in it has no any additional information and can not be unfolded. Property grid for an XML Schema type contains its URI and name.

Part 2 type



Part 2 data view allows access to the upper ontology of ISO 15926-2 lifecycle integration schema. The view has a *filter* box, and all entity types are loaded simultaneously. You will see the full list of ISO 15926-2 types organized alphabetically. This data source is built in the software and is opened as read-only. You can see number of types displayed with current filter and total number of types (201 of course).

Try to type *"arran"* in the *Filter* box.

By expanding the nodes in the tree you can access for any type:

Membership restrictions: node groups *Can classify* and C*an be classified by* contain class membership restrictions as they are defined in ISO 15926-2. Modeling of membership restrictions is based upon the augmented model initially published at https://www.posccaesar.org/wiki/ISO15926inOWLPart2EntityMembership. Some missing relations are added to the Editor data model of Part 2:

- instances of **Class** can classify instances of **Thing**;
- instances of **EnumeratedSetOfClass** can classify instances of **Class**;
- instances of **Scale** can classify instances of **PropertyQuantification**;
- instances of **PropertyRange** or **SinglePropertyDimension** can classify instances of **Property**.

Membership restrictions inheritance is inferred for the type hierarchy (all inferred nodes placed in the same group).

Subtypes and Suprtypes: node groups *Subtypes and Supertypes* allow navigation of a type hierarchy.

Some changes are introduced into the type system to facilitate data verification. Supertype - subtype relationships which are incompatible with OWL model are removed for the following type pairs:

Relationship - **ClassOfRelationshipWithSignature**
ClassOfRelationshipWithSignature - **ClassOfClassOfRelationshipWithSignature**

Roles: node group *roles* contains all roles for relational entities with their type restrictions (EXPRESS attributes in Part 2 representation). Unfolding of nodes in this group allows further exploration of type tree from role restriction type.

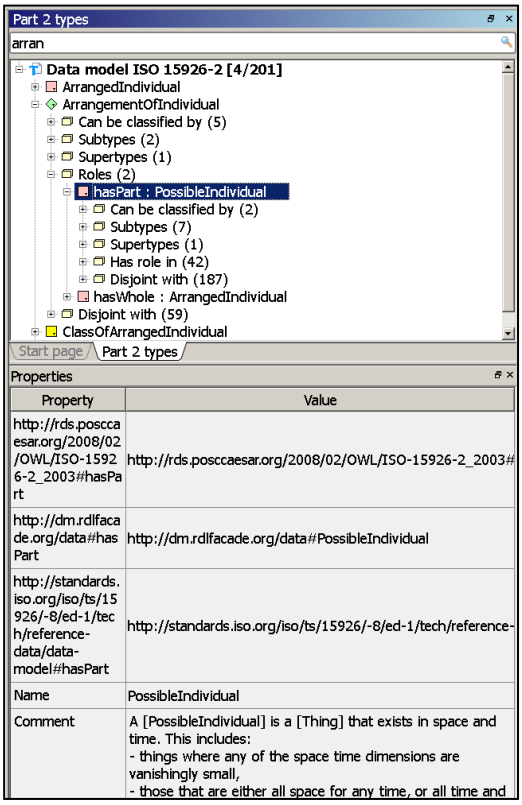
Has role in: node group *has role in* contains all roles where the type is a restriction. Unfolding of nodes in this group allows further exploration of type tree from corresponding relational entity.

Disjoint with: node group *Disjoint with* contains the type all the types it is disjoint with as declared by Part 2 and inferred for the type hierarchy (all inferred nodes placed in the same group).

The model of disjoint types is enhanced to facilitate data verification. Disjoint statements are added for types in the following sets:

Types **Class**, **MultidimensionalObject** and **Relationship** are declared disjoint.

Types **ClassOfClass**, **ClassOfMultidimensionalObject** and **ClassOfRelationship** are declared disjoint.



Property grid for a Part 2 type contains three URIs of this type in three different namespaces: as used in PCA RDL, as were used in retired RDS/WIP, and as defined in ISO 15926-8. Property grid also contains a type name and standard textual definition of a type with notes and examples used in the standard.

Property grid for a role of a Part 2 type contains three pairs of role URI - restriction URI (in the same three namespaces) and description of restricting type. Unfolding the role node you will start exploring its restricting type.

Reference and project data source

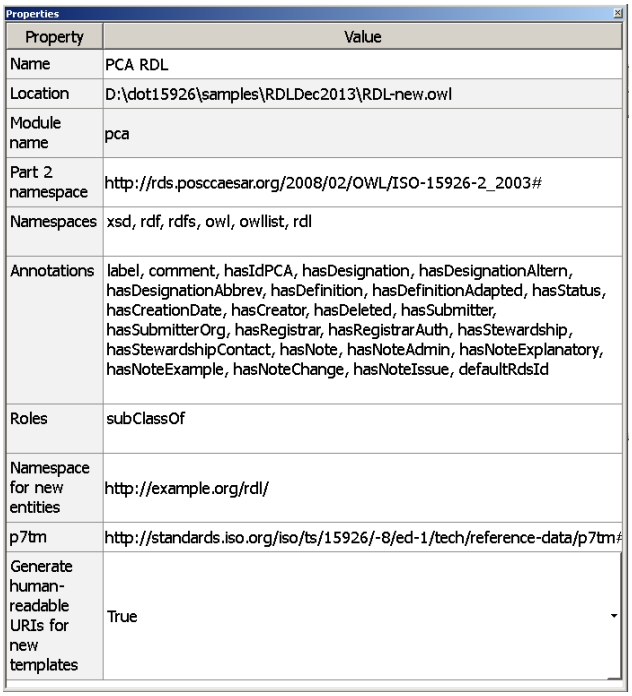
Select the data source in the Project panel or click on the root node in data tree to see property grid for the data source in Properties panel.

Properties of all data sources of the project are saved in a project description file and restored at a next launch or at an opening of a saved project, until data source is removed from the project. Properties of the data source are restored as they were at the moment of removal if data source is restored from the list of recently closed data sources.

Click once on a text field or use double-click to start editing properties with list values. Editing of lists is described in project properties section.

Tab (Shift+Tab) will save your changes and move to the next (previous) value field of a property grid. Click on any other field will also save changes. *Esc* will abandon all unsaved changes made to the field. You can use *Undo* and *Redo* commands after you've saved changes to properties.

Properties of the data source are:



Name – mnemonic name assigned only for convenience of use. When data source is added to the project the name is predefined for built-in or standard sources, and assigned by default rules for all other data sources (as a file name(s) for local sources and an URL for endpoints). *Name* can be edited in a grid (except for data type sets). Meaningful name assignment is useful for navigation in a complex project with many panels open.

Location - data source location (file, multiple files or endpoint URL).

Module name – unique name used for referencing in API function calls of various .15926 Platform components. Module names are defined in project properties.

Part 2 namespace – the namespace used for identification of Part 2 types in a data source. One of three namespaces should be used: PCA RDL, legacy RDS/WIP or ISO 15926-8 (refer to the section Part 2 types above). The namespace can be changed manually; you can copy it from corresponding Part 2 type URI, for example.

The software will try to recognize the Part 2 namespace of a data source, and new data sources are by default created with PCA Part 2 namespace. Please check the namespace before editing, compare it with URI of type identifiers in the data set and change if necessary. The Editor will visualize source with mixed type identifiers (from the three namespaces only), but SearchLan queries will correctly identify entities only in the Part 2 namespace registered in property grid.

Namespaces – the list of namespaces and namespace aliases used in the data source. Aliases are used for URI visualization in data tree. The list is predefined for built-in or standard sources, and standard namespaces *xsd*, *owl*, *rdf* and *rdfs* are added for all data sources used. The software will try to recognize namespaces of newly added data source.

It is possible to edit namespaces in the grid (useful for changing namespace aliases) or add new namespaces. Editing is done by adding, changing or removing space separated pairs **alias namespace** in the list editing form opened with double click. But do it with care!

Annotations –annotation properties (short name and URI) used in the data source. All data sources in the project inherit annotations defined in the project properties, but inherited short names and/or URIs can be overwritten or deleted for a particular data source.

Specific sets of annotation properties for standard files or endpoints are predefined and added by the Editor if standard data source is opened via corresponding menu command. Data sources created by extensions (import from TabLan files or from example catalog JSON files) also have their own specific annotation sets. All other reference and project data sources are opened with default set of annotations which includes ***rdfs:label***, ***rdfs:comment*** and all annotation properties defined in the Part 8 (***annUniqueName, annSynonim, annSource, annNotes***, etc.). Arbitrary

annotations used in a data source are not automatically recognized when it is added to the project. If you know URIs of such annotations and want to facilitate search for them – you should add them to project or to data source properties manually. Editing is done by adding, changing or removing space separated pairs **name URI** in the list editing form opened with double click.

Annotation property ***defaultRdsId*** (with URI *http://posccaesar.org/rdl/defaultRdsId*) is predefined for standard PCA endpoint and PCA RDL file if they are opened through menu command.

Roles – the list of custom object properties (short name and URI) used in the data source. All data sources in the project inherit object properties defined in the project properties, but inherited short names and/or URIs can be overwritten or deleted for a particular data source.

Custom object properties are not defined in Part 2 data model and can be used to record relationships in a non-reified way. Custom object properties, if used in a data source, will not be automatically recognized when it is added. If you know URI of such properties and want to have correct visualization and facilitate search for them – you should add them to project or to data source properties manually. Editing is done by adding, changing or removing space separated pairs **name URI** in the list editing form opened with double click.

The property ***subClassOf*** is inherited as a custom object property from default set of project properties.

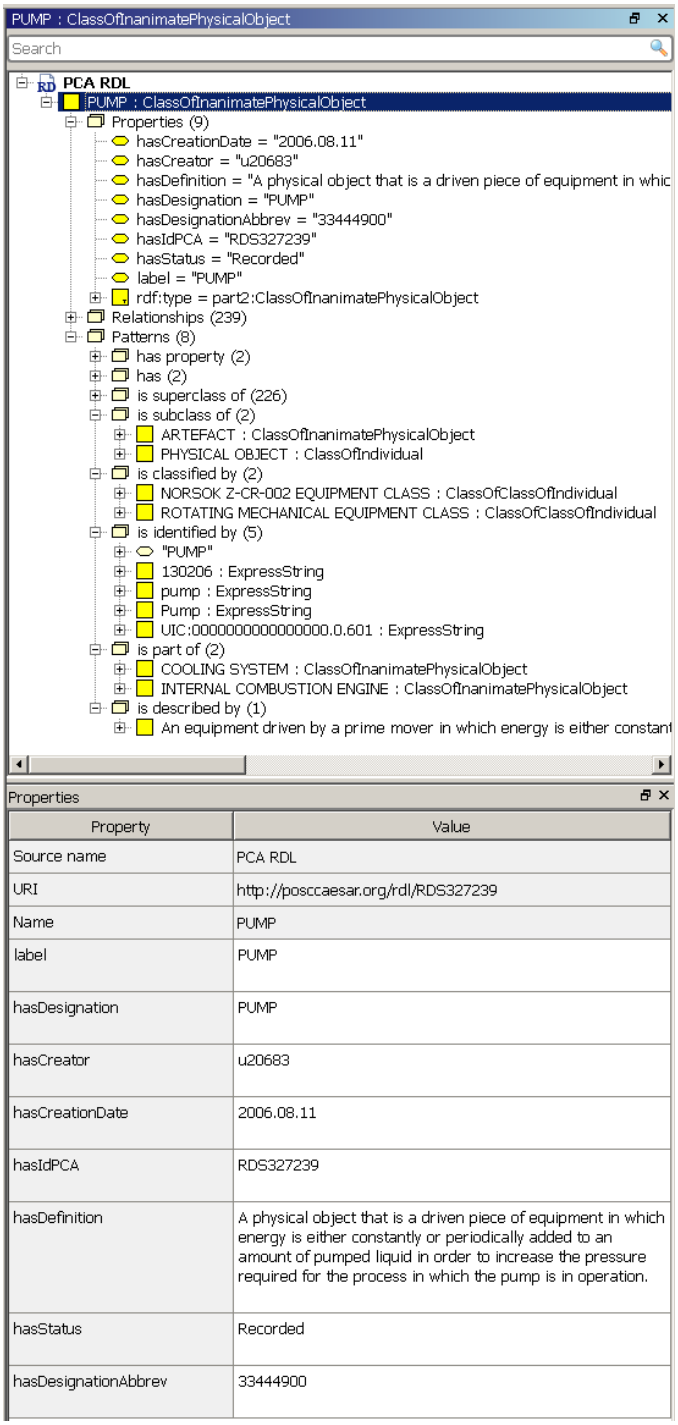
Custom object property ***rdsWipEquivalent*** (with URI *http://posccaesar.org/rdl/rdsWipEquivalent*) is predefined for PCA endpoint if it is opened through menu command *File - Add SPARQL endpoint …- PCA RDL*. It is used to keep links to the legacy RDS/WIP URIs (Rnnnnnn numbers).

Namespace for new entities – the namespace used by default for new entities created in the data source (if URI is not directly specified at entity creation, see below). It is possible to change namespace during the editing and create some entities in a new namespace, but use this option with care, as older namespace will be saved with a non-meaningful alias difficult to recognize later on.

p7tm – namespace for template metamodel ontology used in template definitions if they are contained in the data source. This namespace has changed during Part 8 development and may change in the future. The Editor will recognize namespace used in the loaded data source. For a new data source the Editor uses by default the latest standardized namespace *http://standards.iso.org/iso/ts/15926/-8/ed-1/tech/reference-data/p7tm#*. Namespace can be edited by clicking on the field, but it is not recommended to do so.

Generate human readable URI for new templates – this property has two values. If it is set to ***True***, new templates and new roles created in the data source will receive human-readable URIs by using template and role names as fragment identifiers in namespace defined by *Namespace for new entities* property. It set to ***False*** – URIs for new templates and new roles will be generated as UUIDs compliant to RFC 4122 / ITU-T X.667 / ISO/IEC 9834-8 in the same namespace.

Data entity



Property	Value
Source name	PCA RDL
URI	http://posccaesar.org/rdl/RDS327239
Name	PUMP
label	PUMP
hasDesignation	PUMP
hasCreator	u20683
hasCreationDate	2006.08.11
hasIdPCA	RDS327239
hasDefinition	A physical object that is a driven piece of equipment in which energy is either constantly or periodically added to an amount of pumped liquid in order to increase the pressure required for the process in which the pump is in operation.
hasStatus	Recorded
hasDesignationAbbrev	33444900

A node in a data tree of a data source represents a data entity. Data entity in ISO 15926 compliant data source can be an instance of Part 2 data type, a template definition or a template instance. The type of an entity forms a part of the node name after the colon ":" (except for template definitions). The type is also is reflected in entity icon (see table above).

For an entity the following useful commands are available through *Edit, Search* and *View* menus or through context menu at right mouse click: *Reload item (F5), Search endpoints for URI (F4), Open URI in web browser (F6), Open in new panel (F12).* For command descriptions refer to the corresponding Main menu section.

When unfolded, each entity has *Properties* and *Relationships* node groups. For local data sources there is also *Patterns* node group. Rendering of node groups can be toggled via *View* menu. Special *Simplified entity view* can be turned on hiding all node groups except the content of *Patterns* group.

In terms of RDF model *Properties* group is based on triples where an entity in focus is RDF *subject*, and *Relationships* group is based on triples where in entity in focus is RDF *object*. Group names are chosen to represent ISO 15926 specific usage of RDF. With fully functional RDF viewer/editor at the core, .15926 Editor can parse and present any RDF compliant data set. Visualization of data is specifically tailored for ISO 15926, but analysis of other RDF/OWL data sets is still possible. Please refer to the **Volume 2. APIs of Scanner and Builder** of the documentation and look for section "Non ISO 15926 sources" for more information.

Properties group contains object, literal and annotation properties of an entity.

All entities modeled compliant to ISO 15926 should have an object property *rdf:type* with Part 2 type or template value (other OWL types may be seen in addition to that). Further unfolding of a type node in *Properties* group allows exploration of all instances of this type in the data source, which can take some time. Part 2 type can be explored as data model element by clicking it and pressing *F12.*

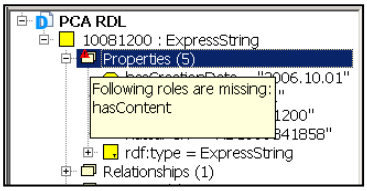
Other object properties include relationship roles and template roles (if this template role has object value). Object role nodes can be unfolded, allowing exploration of role occupiers, if role occupier is from the same data source. If an object role is occupied by an entity from other data

source, unfolding is impossible, but *F12* will open that entity in a separate data view where it can be explored in its original data source context (if available).

If an object property is occupied by an entity which is not available in the project, an icon with question mark "?" followed by its URI will be shown. You can try search on project endpoints with *F4.* Please refer to the section Looking for unrecognized entities to see recommendations for such situation.

Literal properties are properties restricted by some XML Schema type: string, numerical, date or other literal values. Some Part 2 type instances may possess literal properties (*hasContent* property of EXPRESS information representation entity types or values of Cardinality type, for example). Some template instances have literal properties for roles as well (strings or numerical cardinality values, for example).

Annotation properties are literal properties declared as *rdfs:subPropertyOf* of OWL annotation properties according to ISO 15926-8, or used as standard annotation property in a particular RDL. The set of known annotation properties used in the standard data source is described in the data source property grid as described above.



The roles for particular types of entities (Part 2 types or templates) are defined by corresponding data models. Built-in verification on the Editor will attempt to determine whether an entity has missing mandatory roles and whether present roles are proper for this entity type. *Properties* node group and/or roles themselves will be marked with corresponding error or warning signs. Text messages describing an error or a warning will appear if mouse pointer is hovering over the icon. See Data verification for more details.

Relationships node group lists all relationships and template instances in which an entity is playing a role. Each node in *Relationships* group has the form ***roleName* for *relationship***, indicating that entity in focus occupies specific role in relational entity (class of relationship, relationship, template instance, etc). For data source with non-reified relationships (usually non ISO 15926 data source) such wording can be misleading and its correct interpretations requires understanding of RDF triple concept!

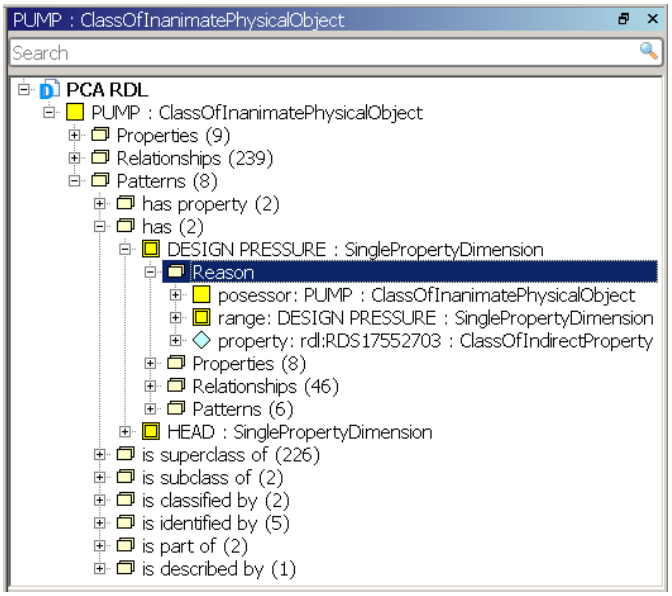
Two commands:

- *Open URI in web browser (F6)* for the role occupier and
- *Open property URI in web browser* for the role itself

are available through *Search* menu or through context menu at right mouse click. For command descriptions refer to the corresponding Main menu section.

Each node in the *Relationships* group can be unfolded to further explore the relational entity itself.

Patterns node group lists all identified patterns – formally defined alternative (but compliant to ISO 15926) ways to express particular relationship between an entity and other entities. *Patterns* group contains subgroup nodes named after the role played by an entity. Each subgroup contains entities which are identified as forming a pattern with explored entity. For example, ***is subclass of*** pattern subgroup contains all identified superclasses of an entity, ***is part of*** subgroup – all entities which are wholes (classes of wholes) for entity in focus.



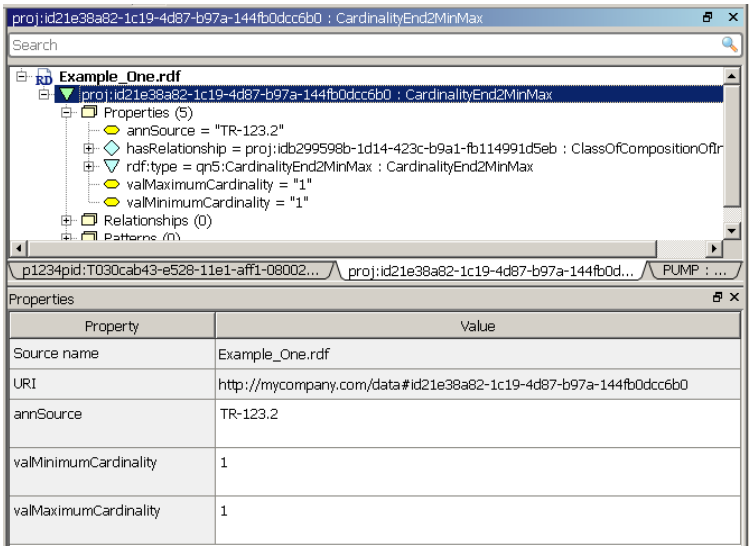
Further unfolding gives an access, in addition to regular data on the entity, to a full data on identified pattern. Special *Reason* node group contains all entities involved in the identified pattern, including pattern signature entities, relational entities (instance of relationship or template) connecting the signature entities (probable by relating them through other entities making up the pattern), and these other entities.

Refer to **Volume 4. Patterns and Mapping** for detailed information about patterns, pattern signatures, direct and inverse role names used in pattern visualization.

Pattern definitions are loaded for data source at the time data source is added to the project. If you've changed pattern definitions, added a new pattern library or added missing template definition module to project – use *File - Reload patterns* menu command (Ctrl+Shift+W). New patterns will be identified for entities. To see new patterns for entities which were already unfolded – use *Reload item* command (*F5*).

Pattern identification and *Pattern* node group visualization can be toggled by the *View - Patterns* menu command *(Ctrl+P).*

Relationships node group for template instance usually has zero elements. *Patterns* node group for template instance can have its type and label identification if corresponding patterns are defined.

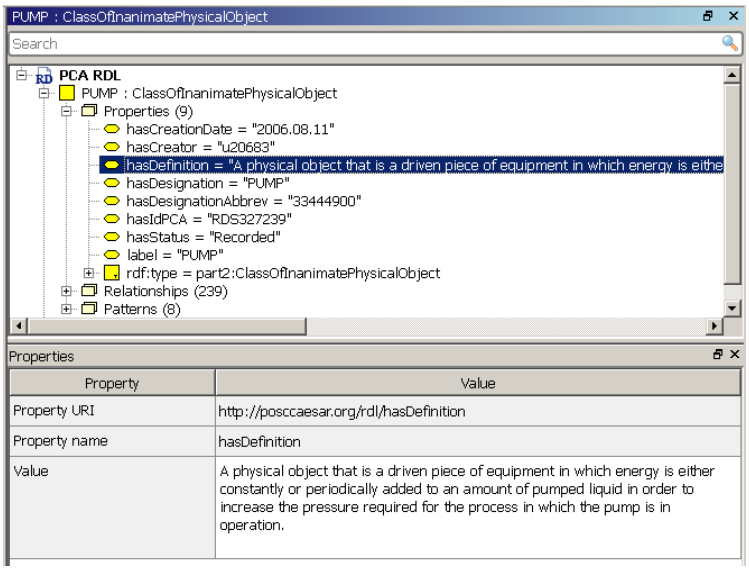


All node groups and subgroups can be turned off by the *View - Simplified entity view command (Ctrl+P)*. In the simplified view only entities which are included in identified patterns for an entity in focus are displayed (with corresponding inverse pattern role labels).

Property grid for reference and project data entity contains the *Source name* with the name of its source, URI and Name of the entity, all non-editable, and also all annotation and literal properties of the entity, all editable. It is impossible to delete a property in the grid, if you delete content of an editable field – this property will get an empty string as a value. To delete a property do it in a *Properties* group in a tree.

Remember that changes to properties of an entity retrieved from an endpoint can only be saved to a local file.

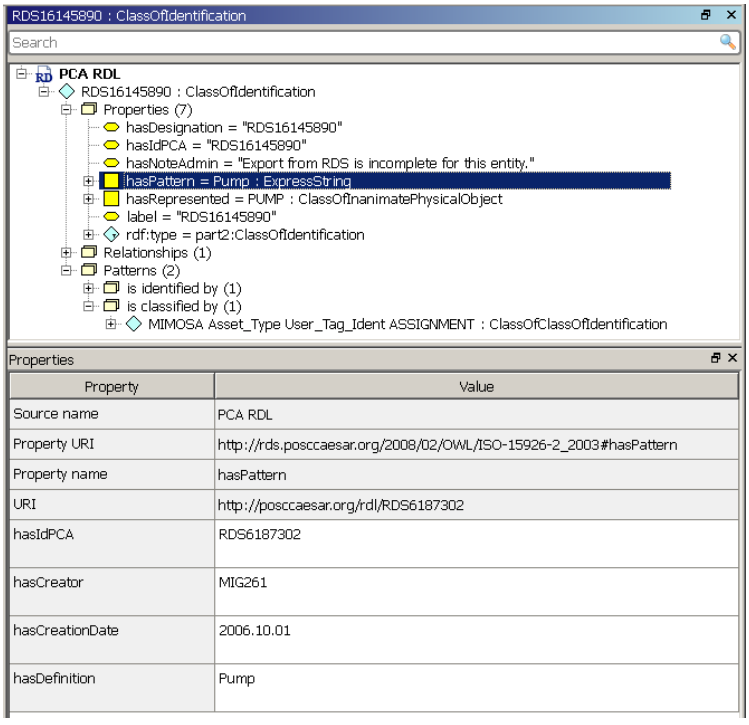
Property of a data entity



A node for an annotation or literal property contains property name (or URI if no short name is registered in *Annotations* of a data source) followed by property value. For an annotation or literal property the command *Open property URI in web browser* is available through *Search* menu or through context menu at right mouse click.

Property grid for an annotation or literal property node contains property URI, property name (URI is used as a name if no short name is registered in *Annotations* of a data source) and property value (editable). It is impossible to delete a property in the grid, if you delete content of an editable field – this property will get an empty string as a value. To delete a property do it in a *Properties* group in a tree.

A node for an object property (role) contains property name (or URI if no short name is available to the Editor through data model, template definitions or *Roles* field in the properties of a data source) followed by a name or URI of role occupier and its Part 2 type.



Two commands:

- *Open URI in web browser (F6)* for the role occupier and
- *Open property URI in web browser* for the property itself.

are available through *Search* menu or through context menu at right mouse click. For command descriptions refer to the corresponding Main menu section.

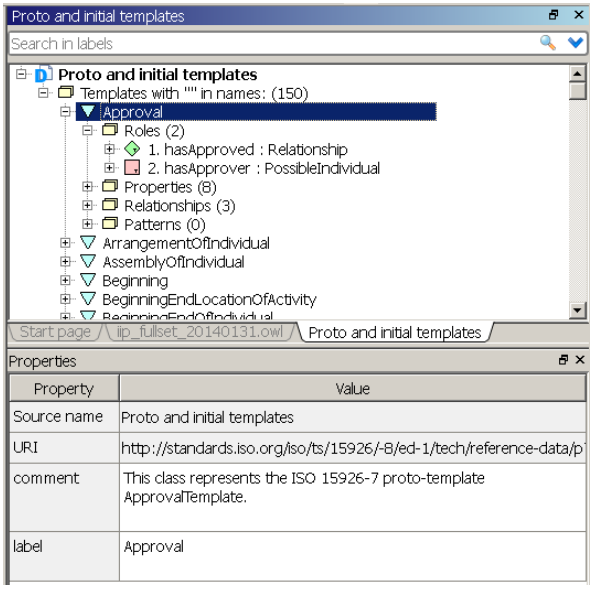
Property grid for an object property node contains the *Source name* with the name of data source, property URI and property name, followed by URI and all properties of an entity which is occupying the role.

For further exploration in the Editor interface the node of object property behaves exactly like the node representing role occupier. You can unfold it (if it comes from the same data source) or open it in a new panel (if it comes from another data source in the project).

Template definition entity

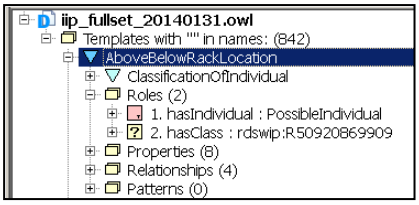
An unfolded template definition node in a data tree of a data source contains (in additional to node groups described above for any entity) a *Roles* node group. Nodes in it represent a signature of an ISO 15926-7 template.

For a template specialized from other template the unfolded node also contains the node for parent template.



Properties and *Relationships* node groups for a template definition contain many specific nodes derived from complex RDF subgraph of template definition. They are mostly marked with ? and it is not advisable to edit these nodes manually.

Property grid for a template has the same fields any other reference and project data entity has: *Source name* with the name of its source, URI and Name of the entity, all non-editable, and also all annotation and literal properties of the entity, all editable.



Template role

A template role node in a data tree of a data source contains role index, role name and role restriction shown after the colon ":" or "=" sign. Role restriction may be by a Part 2 type, or by a reference data entity.

If the role is restricted by value, the role node in the tree contains "=" sign. Restriction by value means that in a specialized template one of the roles of parent template is always occupied by a restricting entity.

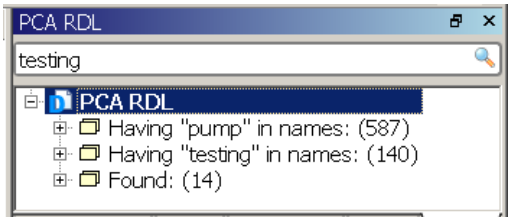
Template role can be unfolded giving an access to the node representing restricting entity. This entity can be further explored either in the context of the current data source, or in the context of its original data source (if it is also present in the project) by opening it in the new panel by pressing *F12.*

Property grid of a template role contains role index, role URI and name, comment, restricting entity and whether restriction is by value (*True*) or not (*False*).

Template role can be opened in a new data panel (press *F12)* as an object or datatype property for exploration in the context of OWL/RDF template definition. In this way annotation properties or classifications of a template role can be explored via its Properties, if they are present in the data source.

Grouping node

Nodes representing groups of other nodes always contain counter of subnodes in the group. If counter is greater then zero – the group can be unfolded. Working with an endpoint be aware that counter for some groups may change for some time as Editor receives more answers from an endpoint.



Some grouping nodes have a predefined name (as *Properties* or *Roles* groups, etc.).

If grouping node contains results of a name search through *search* box at the top of data panel, it has the name *Having "*string*" in names:*

If grouping node contains results of an URI search through *search* box at the top of data panel, it has the name *Search for "*string*":*

If grouping node contains results of an ***All templates from data*** source menu command (executed from ***Search*** menu or from a Toolbar), it has the name *Templates with "" in names:*

The property grid for these kinds of grouping nodes is empty.

If a grouping node was formed as a result of SearchLan query through Pytnon console, the name of a node will be just *Found:* with a counter. The property grid for this node will contain the full text of a query and will allow copying (click with mouse, select and press *Ctrl*+*C*).

The grouping node with search results can be deleted by pressing *Del*. This operation frees memory and can speed up Editor's performance if deleted group contains many data entities. Deletion of search result grouping node does not affect entities in the group.

4.12. Diff view

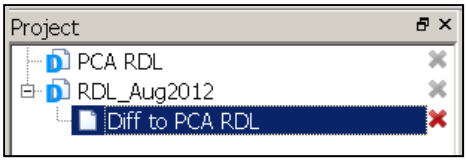
The Editor can build structural diff for two local RDF data sources (including an attempt to identify equivalent blank nodes in RDF graphs). The Editor organizes diff data in a special data view for review and approval. You can save a diff for storage in a versioning system or for exchange with other participants in distributed change management process.

Some theoretical background on comparing RDF graphs and Semantic Web data synchronization can be found in:

- "Delta: an ontology for the distribution of differences between RDF graphs" by Tim Berners-Lee and Dan Connolly, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) http://www.w3.org/DesignIssues/Diff .
- “SEMVERSION: An RDF-based ontology versioning system” by Max Völkel, FZI / Universität Karlsruhe and Tudor Groza, DERI, National University of Ireland, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.84&rep=rep1&type=pdf.

Data comparison available in this version of the Editor is only a starting point for change and configuration management processes for reference and project data which can be supported on .15926 Platform.

4.12.1. Comparing data sources



The Editor allows comparison between two local data sources. Each local data source can be open from one or several local RDF files. To avoid confusion it is recommended to assign unique names to each data source (*Name* field in the property grid for the data source).

In the Project panel click a local data source with right mouse button and navigate to *Compare to…* submenu of a context menu. You will see the list of all other local data sources in the Project. Select any of them for comparison. The Editor will stop responding for some time; duration depends on the size of data sources and number of changes. Please wait till diff view appears on the screen.

Diff will be created and opened as a special data view under the data source selected first in Project panel. This data source can be thought of as "older", and the one selected for comparison can be considered a "newer" one. The diff is built from "older" to "newer" source, and the diff view is placed under the "older" one in Project panel with the name of **Diff to *"newer source name"***.

During the revision process accepted changes will be applied to the "older" data source. If all changes are accepted, the "older" data source becomes structurally identical to a "newer" one.

Of course you can compare data sources which are not really "older" and "newer" versions of each other. Comparison can be useful for data source merging, as Cut-and-Paste process is really complex for large data sets (*Paste* command will either overwrite all existing data about pasted entity in target data source or duplicate properties, refer to Cut, Copy and Paste Data section for more details). If you want to merge two data sources ("target" and "source"), build a diff from "target" to "source" and review all additions and changes in it (see below about filtering additions and changes from deletions).

4.12.2. Diff panel tools

Diff panel toolbar has the following buttons:

Filters:
1. Changed
2. Added
3. Deleted
4. Accept all
5. Save diff
6. Save filtered diff

4.12.3. Reviewing changes

Entities from two data sources are compared to each other if they have the same URI in both data sources (special algorithm is used to identify equivalent blank nodes). Compared data entities are considered unchanged if all their annotation, literal and object properties' values are the same.

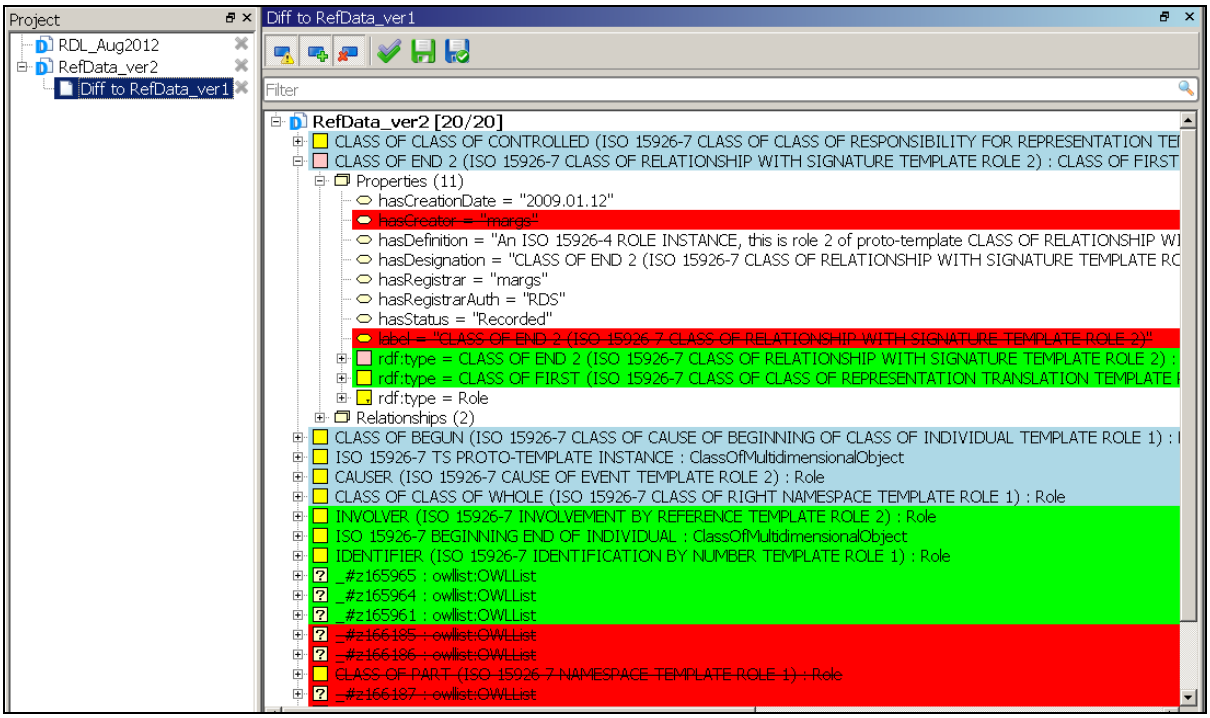
Diff view contains reference data entities which were changed, added or deleted in the "older" data source compared to "newer" one. Entities can be unfolded to view their properties and relationships as it is done in other views (patterns are not shown in diff view).

Added entities are data entities which are present in "newer" data source only. Added entities are highlighted in green. All their properties are also highlighted in green.

Deleted entities are data entities which are present in "older" data source only. Deleted entities are highlighted in red and stricken through. All their properties are also highlighted in red and stricken.

Changed entities are entities which are present in both data sources with the same URI and for which some (or all) annotation, literal or object properties were changed (edited, added or deleted). Changed entities are highlighted in blue. Their Properties group contains added properties in green and deleted properties in red. Changed value of a property is visualized as one added and one deleted property. Properties which are identical in both data sources are displayed for convenience, once and without highlighting.

If an entity has its label changed, it will appear in a diff view as a node with two different names following one another.



Diff view has counter of a total number of entities in it and a counter for filtered view (see below). Entities are shown in increments of 1000 subjects. To add a next group of subjects to the panel go to the end of the list and double-click **...more** line.

4.12.4. Accepting changes

Changes can be accepted individually by double click on an entity or on a property. For accepted change highlighting is removed.

If all property changes of a deleted entity are accepted – deletion of an entity is accepted. If only some property changes of an added entity are accepted – it is added to the "older" data source and becomes changed entity (highlighted in blue).

Changes applied to entities or properties individually can be undone-redone by corresponding buttons.

It is possible to accept all changes with *Accept all* button. In a filtered view only filtered changes (including those that are not yet loaded to the panel) will be affected.

When some changes are accepted, data source from which the diff was built is marked as changed with * before its name. Big volume of accepted changes can not be undone, you will be notified in such cases. You can either save data source to the same or different location, or remove it from the project without saving.

4.12.5. Filtering diff view

Diff view can be quite large – comparison between versions of PCA RDL from April 2012 to March 2013 reveals 74949 entities changed, added or deleted.

To help in a review process Diff view can be filtered in a number of ways:

- via filter buttons on a toolbar, allowing to show changed, added or deleted entities, or any combination of these three categories;

- via *Filter* box at the top of the panel entities can be filtered by name (not by URI) as described in the corresponding documentation section;

- via SearchLan function **show**, which can be used for entities in diff view with queries documented in **Volume 2. APIs of Scanner and Builder** of the documentation.

Filtering by *Filter* box and by **show** filter is applied to entities filtered by buttons. Pressing any filter button will clear all other filters.

New **show** filter or box filter will clear previous **show** filter. But **show** filter can be applied to the results of preceding filtering by *Filter* box.

To clear all filters (except buttons) use empty box filter.

4.12.6. Saving the diff

Structural diff can be saved to an RDF file. **TriG** file format, developed for storage of several named RDF graphs in a single file, is used for diff data. Read "Proposed TriG Specification (the short form)" (Editor's Draft 26 June 2010) at http://www.w3.org/2010/01/Turtle/Trig to learn more about **TriG**.

Diff data is stored as two named graphs (non-URI names are used for test purposes in the current version of the Editor and will be replaced in future versions with semantic markup to support change and version management systems):

- graph named <*deletions*> with all triples present in "older" data source and absent in "newer" data source;

- graph named <*insertions*> with all triples present in "newer" data source and absent in "older" data source.

Notice that it is deleted and inserted triples which are stored in the file. To retrieve information about deleted, added and changed entities these triples should be processed by the Editor!

By pressing *Save diff* button all diff data is saved to a single file.

By pressing *Save filtered diff* button it is possible to save only the data which comprises filtered changes. For example, it is possible to save only data (triples) for changed entities, only for added or deleted, or for a combination of these.

4.12.7. Applying the diff

If you've saved a diff file or received it from somewhere – you can apply it to a data source. Diff can be applied to a data source it was built from (“older” data source) or to any data source for merging purposes, as described above.

Click the data source in Project panel with right mouse button, select *Apply diff...* command from context menu and chose one diff file (in TriG format). Software will stop responding for some time; duration depends on the size of the diff. Please wait till comparison panel with difference data appears on the screen. You can review and accept changes as described above.

If you have saved difference as several diff files (for example filtered by changes, additions and deletions) – you can apply, review and accept them one after another in any order.

If you have a saved diff file and a data source which was comparison target (“newer” data source) – you can apply diff in an inverse way, to get back to “older” data source used in comparison. Click the data source in Project panel with right mouse button, select *Apply inverse diff...* command from context menu and chose one diff file (in TriG format). You can review and accept changes as described above. If all changes are accepted – your data source will become equivalent to the initial data source (the data source that the diff was built from).

Inverse diff application can be useful in reference data library version management. Versioning system can provide access to current RDL version and to the series of incremental diff files for previous versions. It will be possible to obtain any earlier version by applying this diffs inversely.

5. Data editing

5.1. URI and UUID generation

Each time a new entity is created in .15926 Editor, a form appears allowing among other fields manual URI entry. No check is performed whether manually entered URI is unique and well-formed or not. Data entity with URI which is not unique or well-formed can make data source RDF non-compliant and uninterpretable!

If URI field in the form is left blank, the URI will be formed in the namespace defined as a *Namespace for new entities* for the edited data source, using the following fragment identifier:

a. Template name or role name as a fragment identifier for a new template definition (including new specialized template definition) or a new template role, if property *Generate human readable URI for new entries* is set to ***True***.

For example: *http://example.org/tpl#SpecializationOfTesting*

b. Fragment identifier made by concatenating prefix string (with default value "id") with the UUID compliant to RFC 4122 / ITU-T X.667 / ISO/IEC 9834-8 for:

- a new template definition (including new specialized template definition) or a new template role, if property *Generate human readable URI for new entries* is set to ***False***,

- a new instance of Part 2 type or a new template instance.

For example:
http://example.org/tpl#idb9424h86-gjje-kd45-hg24-gd568jgf6778
http://example.org/rdl#idc117f8a9-b305-4fde-b85d-ba27a166889d

To change default value of a prefix string for a particular data source, open it in a data panel and execute command **builder.set_uuid_prefix('*new prefix*')** in the Python console. For example, to make Editor generate RDS-UUIDs in the PCA RDL style execute:

builder.set_uuid_prefix('RDS')

The prefix can be changed only for editable data sources, for read only data sources this command will return an error. You can define different prefixes for different data sources. The prefix will be stored in project description file as data source property. The command *Edit - Put new UUID onto the clipboard (Alt+U)* will generate a UUID with the prefix defined for the data source in active panel.

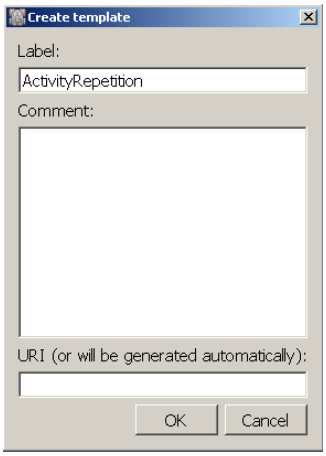
Current version of the Editor does not support automated creation of URIs with human-readable fragment identifiers from labels for Part 2 type instances. The rules for label-to-URI conversion are not determined for ISO 15926 compliant reference data entities.

The editor supports the requirement imposed by JORD project to assign a special annotation property with unique value to all reference data entities, including templates and their roles. This persistent identifier should remain immutable during the life cycle of a reference data entity, which may involve changes in namespace or whole URI change.

To do this a special annotation property with short name **defaultRdsId** should be added to project Annotations or data source Annotations. For PCA RDL and federated data libraries this property has now URI **http://posccaesar.org/rdl/defaultRdsId .** It will be added by default to all new projects created in the Editor. If URI (not a short name!) is changed – all new entities will receive the annotation with new property (remember that annotation defined for a data source has the precedence over the annotation for the project). If **defaultRdsId** property is absent from both data source and project – new entities will not receive it.

Value for the **defaultRdsId** is defined exactly by the rules defined for UUID as URI fragment identifier (including prefix, described above). For all entities created with UUID as a part of URI – **defaultRdsId** value will be the same as fragment identifier.

5.2. New template



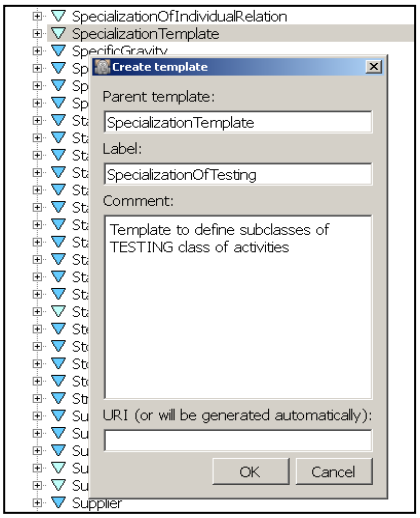
A new template can be added to any data source.

To add a base template click the data source name root node and press *Shift+Ins* or use *Edit – Add template* menu command. A form will appear allowing to enter template name, to fill Comment field (description and/or axiom) and enter an URI (optionally).

Additional annotation properties and specializations can be added to the template in the same way they are added to other reference data items: by pressing *Ins* on a *Properties* node or by dropping superclass on this node and selecting *subClassOf* object property.

Pressing *Del* deletes the template. It is impossible delete a template if there are templates specialized from it. The Editor will ask you whether you want to delete all specialized templates together with parent template.

5.3. New specialized template



To create a specialized template click a parent template and press *Shift+Ins* or use *Edit – Add template* menu command. Specialized templates can be created only in the same data source with parent!

In the form for specialized template the same fields as for a new template can be filled. Additional annotation properties and specializations can be added to the template in the same way they are added to other reference data items.

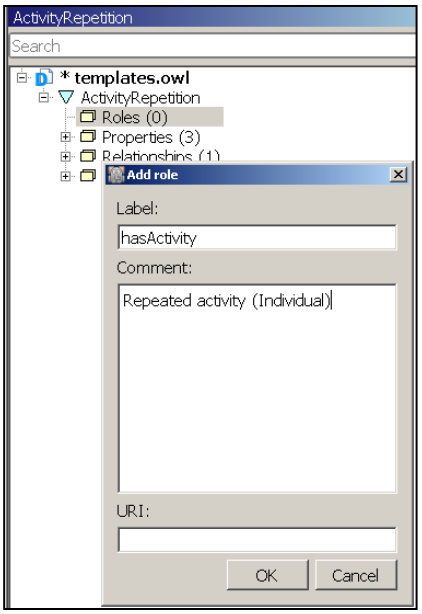
5.4. New template role

New template is created without roles (with the exception of specialized templates who inherit parent roles). Pressing *Ins* on *Roles* node of a template brings up a form to create a new role. Role label and description can be entered, together with URI (optionally).

Roles in different templates but with the same label will get the same URI (human-readable or not). It will be impossible to assign different labels to such roles later.

New template role is always created with *xsd:string* type restriction. You can change restriction later using XML Schema types, 15926-2 types or RDL entity (see below).

When you add a new role to a parent template or delete the role – the change is applied to all templates specialized from it. When you add a new role to a specialized template or delete a role – this change is applied to its parent template and propagated to all templates specialized from it. Thus it is impossible to change number of roles of specialized template compared to parent.



Pressing *Del* deletes the template role.

5.5. Role editing

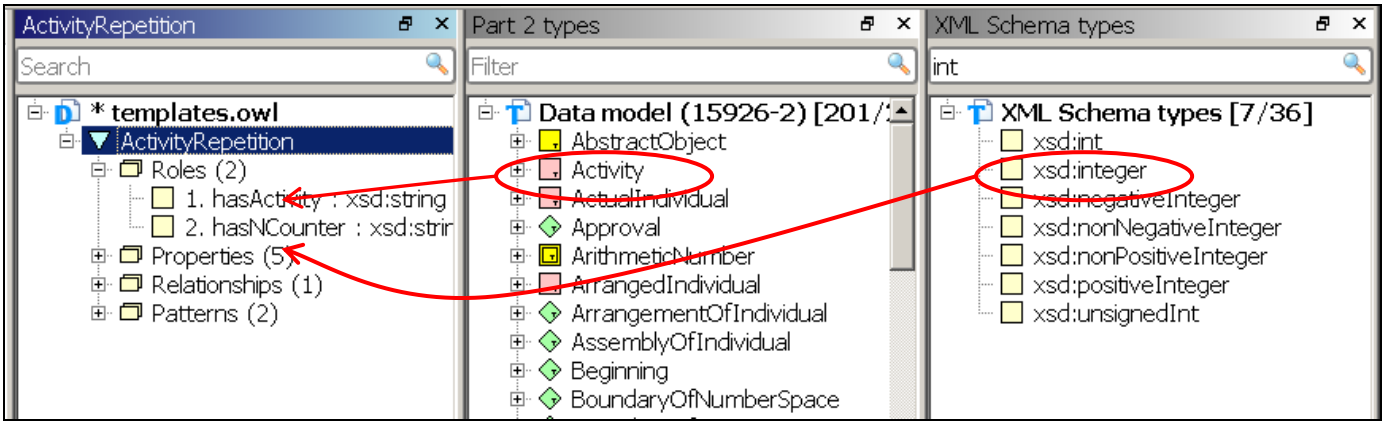
The role name can be edited in the role property grid.

Role order (index) is determined by the sequence of roles in the tree. Role order can be changed by rearranging roles with mouse dragging.

When you edit a role name or rearrange role order the change is propagated from parent template to all templates specialized from it; and from specialized template to its parent template and then to all templates specialized from it. Thus it is impossible to change role names or role order of specialized template compared to parent.

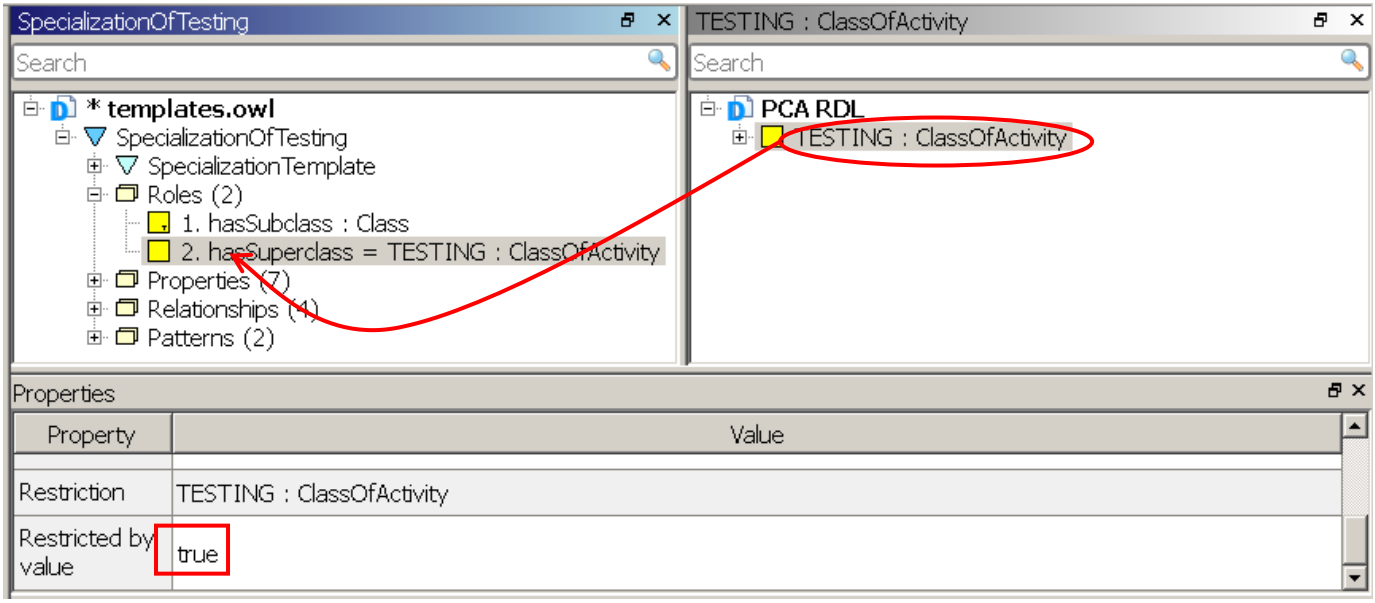
Role name change is also propagated to all templates which use the same role (the OWL property with the same URI).

To change restriction of a template role open Part 2, XML Schema data source or any reference and project data source in a new panel, drag the required type or entity and drop it on a template role. Restriction of the role will change.



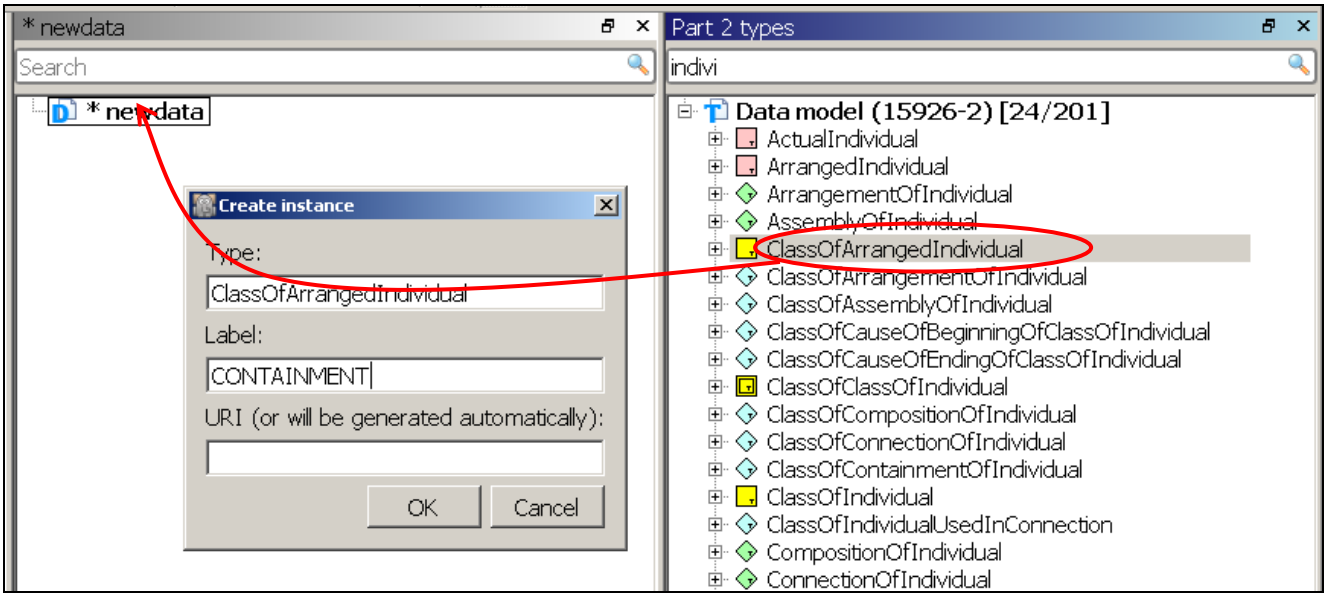
If restriction by value is required, change *Restricted by value field* in the property grid of a role to *True*. Restriction by value is marked by the "=" sign in the role description.

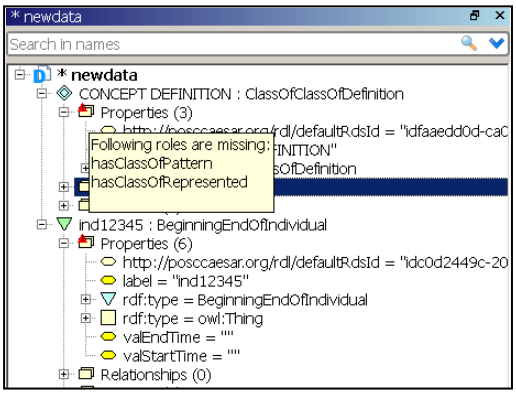
Restriction changes do not propagate from specialized templates to parent or back. You have to ensure that role restriction for a specialised template (by value or not by value) meet the restriction of a corresponding role of a parent template. Published version of the Editor does not enforce this rule during editing and does not do this type of verification yet.



5.6. New data entity

To create a new instance of Part 2 type open a Part 2 data source next to the panel with edited data source. Drag the required type and drop it on the root node of the edited data source. A form will appear indicating the Part 2 type and allowing entering the name of a new entity and URI – both optional. An entity without a name can be assigned identification later by adding *label* property, other annotation property, or in some other way allowed by ISO 15926 (relationship or template instance).





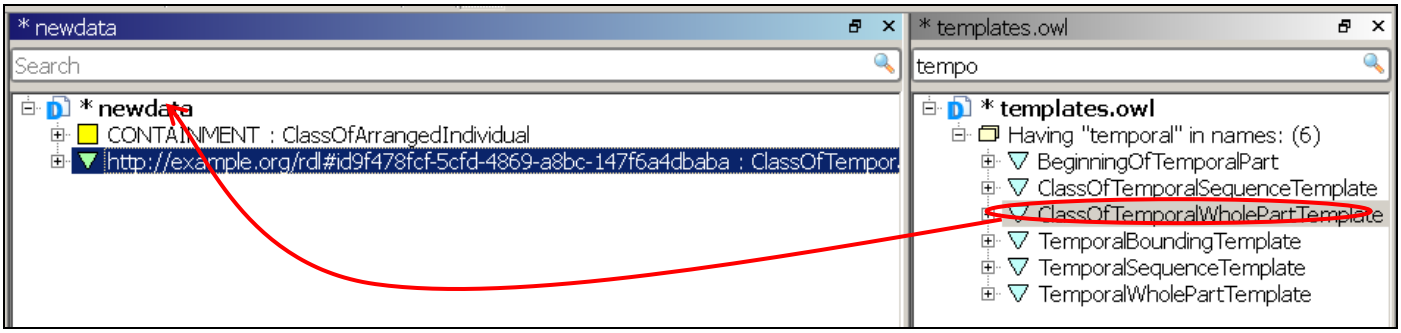
To create a new template instance open a view on a data source with template definition next to the panel with edited data source. Drag the required template and drop it on the root node of the edited data source. A form will appear indicating the template type and allowing entering the name of a new instance and URI – both optional. Template instances aren't usually assigned proper names.

All template roles are mandatory for template instance, and there are mandatory roles for some instances of relational Part 2 types. When an instance with mandatory role is

created – a sign will appear at the *Properties* node group indicating that mandatory roles are missing. The names of missing roles can be seen if mouse pointer is hovering over this node.

Relational type instances are created without roles (object properties), except roles restricted by value in the instances of specialized templates and literal template roles. Roles restricted by value in the instances of specialized templates are filled with restricting values in place. Literal template roles are created with empty string values. Therefore they are not marked as missing at the time of creation, although their values are not properly defined yet.

Please refer to the next section for role filling.



A type of an entity (Part 2 type or type of a template instance) can be changed by dropping new type (Part 2 or template definition respectively) on the *rdf:type* node in *Properties* node group of the entity. This can lead to exotic constructs (classes of individuals with relationship or template roles, relationships or templates with wrong role names, etc.) so the editor will try to determine wrong roles and warn you about them.

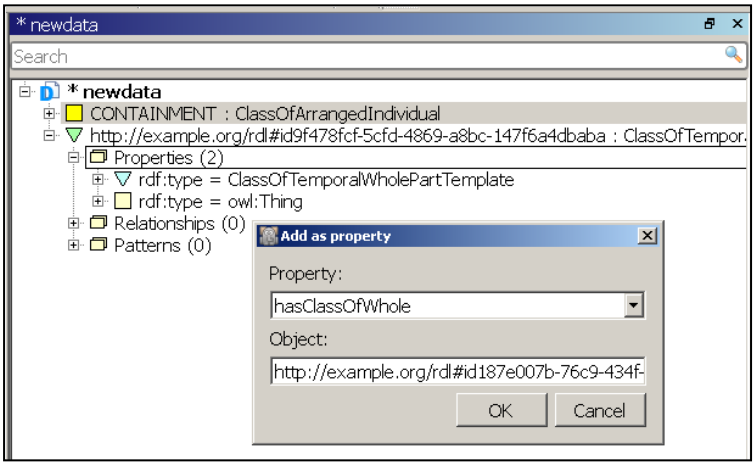
Object roles defined for Part 2 entity types can appear only for instances of corresponding types, if they are met for instances of some other type they'll be marked with error sign. Template roles used for other template instances will be marked with warning sign only. Refer to Data verification section for more details.

Pressing *Del* deletes the entity.

5.7. Filling the object property (role)

To fill an object property (role) of a relational entity (instance of a relationship, of a class of relationship or template instance) drag the entity you need in a role to *Properties* node of a relational entity.

A role occupier can come from the same data source or from any other reference and project data source present in the project, file or endpoint. The link and cross-navigation possibility will remain in place as long as both data sources remain in the project.



After the drop a form will appear to choose object property (role name) from those available for the type of edited relational entity: roles for relationships or for classes of relationship, template roles, *rdf:type* and custom object properties.

If you are creating and filling a new role, be careful to drop an entity on a *Properties*

grouping node itself, not on one of existing object property nodes in the group! An entity dropped on the existing property can change the role occupier.

Notice that ***rdf:type*** and all object properties registered as *Roles* (custom object properties) in the project properties and in the properties of a data source are added to the list of available object properties. If properties with the same short name but with different URIs are defined for project and for data source – URI defined for the data source will be used.

Custom object properties and ***rdf:type*** will appear even if an entity is dropped to *Properties* node of a **non-relational** entity. This feature allows to use ***rdf:type*** object property to represent classification relationship between data entities or use ***rdfs:subClassOf*** property to represent specializations. This is allowed by Part 8 but rarely used now. It is also possible to use custom object properties to record other relationships in a non-reified way.

It is possible to add second Part 2 type to an entity or create multiple instances of the same role in a relational entity, The Editor will mark duplicate roles as errors. The Editor will mark the *Properties* group with an error if a mandatory role is missing.

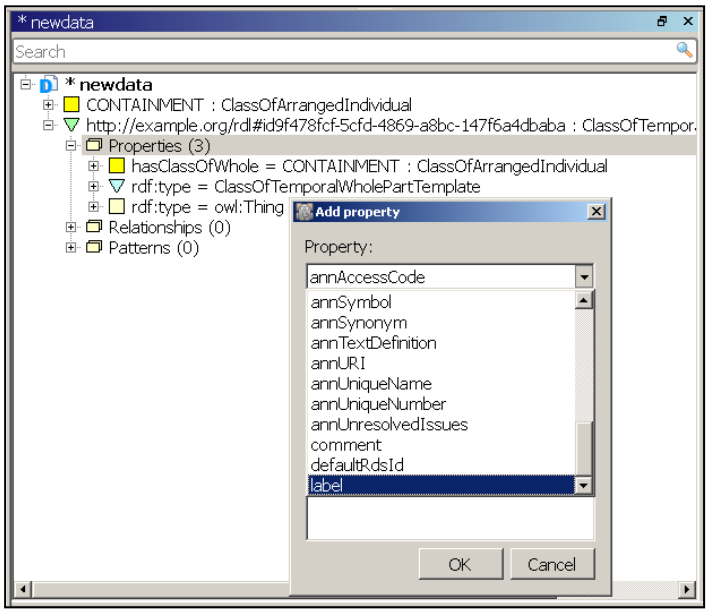
Instances of specialized templates with object roles restricted by value are created with corresponding role already filled with proper entity, but it is possible to change this role occupier later. Please avoid such changes. The Editor will mark specialized template instance roles with wrong values as errors. Refer to Data verification section for more details.

Pressing *Del* deletes the object property role.

5.8. Adding literal or annotation property

Some Part 2 type instances may possess literal properties (for example, Cardinality instances may have literals *hasMaximumCardinality* and *hasMinimumCardinality*, instances of six subtypes of ClassOfExpressInformationRepresentation may have *hasContent* literal property). Template instances can also possess roles with literal values, to be filled with numbers, strings, dates and times, etc. (for example, template BeginningOfTemporalPart has role *valStartTime* restricted by type *xsd:dateTime*, template CardinalityEnd1MinMax has roles *valMaximumCardinality* and *valMinimumCardinality* restricted by type *xsd:double*).

Template instances are created with literal roles already in place but with empty value. The value can be added in a property grid of an instance or of a role itself.



Annotation properties may be added directly to *Properties* node group of all Part 2 type instances and all template instances of a data source. To add literal or annotation properties to an entity click its *Properties* grouping node and press *Ins* or execute *Edit – Insert new item* menu command. The form will appear allowing to choose required property and set its value. The list of available properties is formed from the following sources:

- set of annotations defined for the project (to extend this set refer to the project properties description);

- set of annotations defined for the data source, (to extend this set refer to the Annotations in data source property grid description);

- literal properties defined for a type of entity edited (literal roles for some Part 2 types and literal roles for templates) – look to the end of the list to see these.

No type control for values assigned is provided by the Editor. The Editor will control only whether mandatory roles are filled or not and mark as mistakes missing mandatory roles (on the *Properties node)* and duplicate literal properties (but not annotations).

You can edit existing literal and annotation property value in a property grid of an entity or of a property (white fields in a Properties panel allow editing).

Pressing *Del* deletes the property.

5.9. Cut, copy and paste data

It is possible to move data between data sources with *Copy (Ctrl+C), Cut (Ctrl+X)* and *Paste (Ctrl+V)* commands available from *Edit* menu, context menu, or via keyboard shortcuts. As ISO 15926 RDF data sources are not text files, this set of common commands is augmented with *Copy text (Ctrl+Shift+C)* and *Paste as triples (Ctrl+Shift+V)* commands. Selection of available commands is context-aware and their execution is subject to some restrictions, as described below.

5.9.1. Data selection

You can select a single entity, an individual property, or several entities and/or properties. Multiple selections are done with mouse clicks while simultaneously holding *Shift* or *Ctrl* key.

Selection of a grouping node *Properties* is equivalent to the selection of corresponding entity.

Grouping nodes *Relationships* or *Patterns* can be included in group selection, but *Copy* and *Cut* commands have no effect on them (except *Copy text*).

You can also select a search result grouping node in a data tree containing several entities. *Copy* and *Cut* commands will be applied to all entities in such a group.

For a template definition any selection of templates and/or roles is possible, but available commands will depend on the selection made.

5.9.2. Data copying

You can *Copy* entities or properties in any reference and project data source (local or endpoint).

You can *Copy* whole template definitions (not roles) in any data source (local or endpoint). Be careful while copying specialized templates – correct paste to other data source is possible only if parent template is also copied to new data source.

You can *Cut* entities or properties in any local reference and project data source.

You can *Cut* whole template definitions (not roles) in any local data source. You can not *Cut* a template definition without deletion of all templates specialized from it. The Editor will ask you for confirmation. **If confirmed – all specialized templates will be just deleted, only parent template will be placed in the clipboard!**

When you execute *Copy* or *Cut* commands RDF data (one triple, several triples or complex template subgraphs) for selected entities or properties are placed in the Editor's clipboard. Notification with the number of triples copied appears and is stored in Console history area.

Additionally URIs for all selected entities (subjects in all copied triples) are placed in a standard Windows clipboard.

For any selection of entities, properties and groups special command *Copy text (Ctrl+Shift+C)* is available (standard *Copy* may be disabled for a group of diverse elements). *Copy text* command copies visible text of selected tree nodes to a standard Windows clipboard. Tree structure is preserved with ***Tab*** symbols. This command is useful for written communication about data entities.

5.9.3. Data pasting

To use *Paste* from menu just move focus to the required data source. To *Paste* something from context menu right-click anywhere on a data source view. It doesn't matter whether you click a tree element or an empty space. You can *Paste* entities or template definitions only to a local data source. Be careful while pasting specialized templates – correct paste is possible only if parent template is also available in a new data source.

It is possible that you are pasting to a data source some data about entities which are already present in it (there are triples with the same subjects in both data source and Editor's clipboard). There are two *Paste* commands in the Editor with different behavior in this situation.

Regular *Paste (Ctrl+V)* command will assume that you are moving new entity to a data source in its entirety. Therefore it will ask you whether you want to overwrite existing information about this entity. If confirmed, all information in the data source about an entity will be replaced with new information (all existing triples with this subject will be deleted and replaced with triples from the clipboard). If you are pasting many entities you will be offered this choice for each entity or you can confirm some choice for all further requests.

If you really want to augment existing information, use *Paste as triples (Ctrl+Shift+V)* command. It will add unconditionally all triples with the existing subject and any new predicate. If triples with the same subject and predicate (but different objects) are found in the data source and in the Editor's clipboard (if such property for this entity already exists with another value), the Editor will offer you a choice:

- to replace an old value with a new value;
- to add new value in addition to the old (some properties can have multiple values);
- to keep an old value and skip pasted data.

If you are pasting many entities you will be offered this choice for each triple or you can confirm some choice for all further requests.

Data from Windows clipboard can be pasted to other applications or to Console input area (useful for searches by URI). URI(s) will be pasted if *Copy* command was previously executed, and text will be pasted if previous command was *Copy as text.*

6. Data verification

Data verification for ISO 15926 is a complex issue. There are numerous ontologies which may impose restrictions on data. Data model for ISO 15926 data as defined by Part 2 is one such ontology. Each template definition data source defines its own ontology and imposes its restrictions. Various domain ontologies are distributed among federated reference data libraries. There are some hopes that uniform OWL representation will allow the use of general-purpose reasoning algorithms to do the verification. Another approach is to use some query language to construct verification rules for specific situations. There are attempts to use SPARQL for this task.

.15926 Editor has built-in verification tests with rules implemented using SearchLan – search API of Scanner module. These rules are based on patterns to abstract from specific data representations used in ISO 15926 data modelling (reified relationships, RDF predicates, templates). More on SearchLan and patterns can be found in **Volume 2** and **Volume 3** of this documentation.

The set of built-in tests will grow with future releases of the software.

Verification rules are applied in real time as data are added or edited in the data source via Editor's GUI. Verification of the whole local data source can be executed by **Search – Search for suspicious entities** menu command or by console command **show(id=wrong)**. Be aware that verification of a large data source can take significant time (hours for PCA RDL)! Console commands allow to do verification for smaller subsets of data, refer to **Volume 2 APIs of Scanner and Builder** for details.

You'll probable see more warning messages then errors in your verification results. It is because of the so called "open world assumption" at the core of ISO 15926 ontology. Under this assumption many suspicious relationships between entities can not be judged outright false using only limited knowledge (set of facts available in present data sources). Each warning message should be investigated carefully.

6.1. Mandatory and optional roles verification

Mandatory and optional roles for relational entities in ISO 15926 data are defined:

- by Part 2 data model for instances of Part 2 entity types;

- by each template definition data source for template instances (in fact all template roles are mandatory).

The Editor does verification for:

- missing mandatory roles;

- duplicate roles;

- misplaced roles (roles from one data type or template appearing for an instance of different data type or template).

While restrictions on Part 2 role usage are rather strict, restrictions for template role usage are more relaxed, and role predicates theoretically can be used not only in the context of templates. Therefore built-in data verification in the Editor handles misplaces Part 2 and template roles differently. See the table below for errors/warnings for different situations.

	Missing mandatory object role	Misplaced P2 object role	Misplaced template object role	Missing mandatory literal role	Misplaced P2 literal role	Misplaced template literal role
Instance of Part 2 type	Error	Error	None	Error	Warning	None
Instance of template	Error	Error	Warning	Error	Error	Warning

6.2. Typing and classification verification

In ISO 15926 data an entity may be declared as belonging to some class in a number of ways. An **rdf:type** predicate and instance of **Classification** type are only the top of the iceberg. A number of templates can be used to declare classification, classification can be inferred from a chain jf specialization relationships which in turn may be declared in a number of ways.

To collect as much statement about classification as possible, we are using patterns Classification and Specialization defined in the Editor's pattern libraries. The Editor is distributed with a set of example patterns which includes the most used ways to express there relations. Other pattern libraries which we will distribute may expand this set, including expansion of Classification and Specialization patterns. Refer to **Volume 4. Patterns and Mapping** for more details.

The set of classifiers for an entity is determined as all classes which form Classification pattern with this entity (as classifiers) and all classes which form Specialization pattern with them (as superclasses)

Current version of the Editor doesn't test whether this set of Classifications is non-contradictory!

We are starting the set of typing and classification tests in the Editor from verification of typing for relational entities – instances of relational Part 2 types and instances of templates.

Each relational Part 2 type and each template has role restrictions – roles can be restricted with Part 2 types for Part 2 types and base templates, and with classes for specialized templates. For all entities filling the role in an instance of a relational type or in a template instance the role restriction is checked against classifier set determined as described above.

If role restriction is found among the classifiers of an entity – verification test is passed.

If role restriction is disjoint with one of the classifiers of an entity – verification test fails and role icon is marked with error sign (a message explaining an error is displayed if mouse pointer is hovering over a role). Notice that explicit disjoint statements are available only for data model types.

In all other cases appears a warning sign.

6.3. Role value verification

Specialized templates may have roles restricted by value. No other entity can appear in such a role. The Editor will show an error sign if a role restricted by value is occupied by an entity different from the one defined in the template.

7. Walkthrough guides

7.1. Part 4

Part 4 file is a representation of Part 4 tables in a legacy RDF format and can be downloaded from http://rds.posccaesar.org/2009/08/OWL/ISO-15926-4_2007. It can be added to the Editor environment by selecting *File – Add file(s)… – Part 4 file…* menu command (you can register this file via menu *File – Settings*, *Paths* tab). If opened in this way, the data source is loaded with customized set of annotation properties. Select the data source Part 4 in Project panel and open it in a new data panel (*F10*).

You can navigate through the file which has 9566 named entities. You can load them all with an empty search in the *search* box or look there for “pump”. Navigation allows you to move between data sources. Try selecting the Part 2 type of any entity (*rdf:type* in *Properties* node group) and press *F12.*

Original ISO 15926-4 was distributed in the form of .xls spreadsheets corresponding to data modeling and design domains, the spreadsheet names can be found at the page http://rds.posccaesar.org/2008/05/XML/ISO-15926-4_2007/ . You can search for items from a particular spreadsheet with a SearchLan console query for annotation property *spreadsheet* structured as in the following example:

show(spreadsheet=('http://www.tc184-sc4.org/ts/15926/-4/ed-1/tech/rdl/electrical.xls'))

Notice that Part 4 entities from this dataset are not included in the PCA RDL, Part 4 data is reproduced there with numerous changes and additions.

7.2. PCA RDL

Details about PCA reference data library and services can be found at https://www.posccaesar.org/wiki/Rds .

Endpoint content is available for download from http://rds.posccaesar.org/downloads/PCA-RDL.owl.zip (the file is sometimes updated, use the newest version available). Download and unzip the file, and register the path to it through *File-Settings* menu command (*Paths* tab, *PCA RDL (RDL.owl)* field). Add the data source via *File – Add file(s)…– PCA RDL file…* menu command. The file is relatively big (more then 57 000 entities, almost 3 mil. triples), so you will see loading progress. You can continue work with the program while big datasets are loaded. Open the data source in a new panel when loading is complete.

Type "*celsius*" in the *search* box and press *Enter*. Explore properties of entities found. Selecting entity node in the tree you can see its properties in the Property panel.

Now you can go to *File-Add SPARQL endpoint…* menu and chose PCA RDL endpoint.

Search for "*celsius*" again. You will see the same reference data entities in online version of PCA RDL. You can press *F6* for items from local file or from an endpoint and see dereferenced URI in your web browser.

Open *Relationships* node group for any entity from an endpoint and find a relationship with a question mark icon. PCA RDL contains references to the old RDS/WIP reference data library (now retired), as RDS/WIP URIs are used in many legacy data sets and mappings. These references

are represented through the use of special RDF predicate *http://posccaesar.org/rdl/rdsWipEquivalent.* Such representation has no semantic meaning under ISO 15926 modeling conventions and is not compliant with ISO 15926-8 rules for RDF/OWL representation.

7.3. Looking for unrecognized entities

Sometimes exploring a data source in .15926 Editor you meet an unrecognized entity in a relational entity role, in a template role, or as a value of some object property (probable also unrecognized). Unrecognized entity looks like an icon with question mark "?" followed by URI or by a role/relation name and URI. URIs in the tree are usually prefixed by namespace aliases not easily recognizable. Click on the entity node and look at the property grid to see a full URI.

> To study examples described below – make sure that Project panel is empty and add reference and project data file ***sample_lookup.rdf*** from <*samples*>\ folder. Open it and do an Empty search (hit *Enter* in the *search* box). Expand found instances of **ClassOfAssemblyOfIndividual** and of **Specialization** to see unrecognized role occupiers in them.

Look whether you can recognize data sources in which these entities may be found. Look for unrecognized entities at endpoints or just load local copies of RDLs. Then try to reload (*F5*) unrecognized entity.

> In our example both unrecognized entities in roles of *assembly1* instance are in the namespace *http://posccaesar.org/rdl/* indicating that they are from PCA RDL and you can look it up in local export file or on an endpoint.
>
> First add PCA endpoint to the project via *File-Add SPARQL endpoint…- PCA RDL*, right-click hasClassOfPart role and choose *Search endpoints for URI* command (*F4*). Found item will appear as a separate view under PCA RDL SPARQL node in a Project panel. Reload (*F5*) *assembly1* entity. One role occupier is recognized.
>
> The second role occupier we can identify via the same search, or we can find it in local copy of PCA RDL. Remove endpoint from the project and use *File-Add file(s)…–PCA RDL file…* menu command. No search is required as items from local source are linked in the project immediately. Just reload (*F5*) *assembly1* entity and see both role occupiers. You can click on them and press *F12* to explore these entities in their native data source.

Check whether unrecognized entity has an URI looking like a legacy RDS/WIP URI: *http://rdl.rdlfacade.org/data#* namespace and fragment identifier starting with *R* and followed by numbers or UUID (called "R-number"), for example *http://rdl.rdlfacade.org/data#R60717709462*). To identify an entity behind such URI try *Search – Search PCA endpoint for WIP equivalent* (*F7* or corresponding toolbar button) menu command. It will open PCA endpoint and search it for an equivalent class recorded there by special relationship *rdsWipEquivalent*. Search results are added as a new data view for PCA endpoint data source in Project panel.

> In our example click *hasSuperclass* role in *specialization1* instance. It is occupied by h*ttp://rdl.rdlfacade.org/data#R35802804974*. Hit *F7* button. See PCA RDL endpoint and one new view for it appearing in Project panel. Open new view and look at the found entity PHYSICAL OBJECT. You can find in its Relationships group the following relationship to the URI we were searching:
>
> *hasWipEquivalent for* h*ttp://rdl.rdlfacade.org/data#R35802804974*

Even if search for WIP equivalent is successful – the entity will remain unrecognized in your data source, as equivalence is recorded in PCA endpoint in a way which has no semantic meaning under ISO 15926 conventions. You can edit your data source – drag newly found entity and drop it on the property with unrecognized URI. After that you can save snapshot of an endpoint and keep the snapshot file in your project. If you prefer to use endpoint data, you can next time find unrecognized entity (with new URI!) through *Edit – Search endpoints for URI* (*F4* or toolbar button) menu command described below.

> Drag PHYSICAL OBJECT from endpoint panel and drop it on *hasSuperclass* role in *specialization1* instance.

Be aware that many URIs in RDS/WIP namespace were issued for various proprietary datasets and project sandboxes, and were never registered in RDS/WIP RDL. Such entities of course have no equivalence relationship recorded in PCA RDL. One other place to look for such entities in iRING sandbox at http://www.iringsandbox.org/repositories/SandboxPt8/query . You can add it to your project via *File-Add SPARQL endpoint…- iRING Sandbox* menu command.

If your unrecognized entity has some other URI, or an equivalent for WIP-looking URI was not discovered in PCA RDL or iRING sandbox, you can try search on other endpoints. Add all endpoints you want to search to the Project panel.

When all required endpoints are added to Project panel, click an unrecognized URI and use *Edit – Search endpoints for URI* (*F4* or corresponding toolbar button) menu command. Search results are added as new data views for each SPARQL data source in Project panel. If you have found required URI - reload (*F5*) an entity with unrecognized property.

> Search for RDS/WIP equivalent (*F7* button) will not deliver any results for an entity occupying *hasSublass* role in *specialization1* instance. Add iRING Sandbox to the project as described above, click *hasSublass* role in *specialization1* instance and press *F4.* Reload *specialization1* instance (*F5).*

Remember that when you restart the Editor next time, the search of external endpoints will not be repeated. You can save snapshots of required endpoints and add local files to your project, and then all linked reference data will be reloaded after the restart.

7.4. Exploring endpoints

Open **IIP Sandbox (templates)** from ***File – Add SPARQL endpoint…*** menu. Press *A**ll templates from data source*** button on the toolbar and look at roles of identified specialized templates. You will see many roles restricted by unrecognized entities in the namespace h*ttp://rdl.rdlfacade.org/data# .* This is legacy namespace of RDS/WIP reference data library. Today RDS/WIP reference data library URIs are preserved as values of *rdsWipEquivalent* predicate in PCA RDL.

The same namespace was used also in a number of other sandboxes, including iRING Sandbox (*http://www.iringsandbox.org/repositories/SandboxPt8/query*) which keeps reference data for IIP team (at some point in the future its content will be moved to PCA RDL).

Add PCA RDL and iRING Sandbox endpoints to your project via ***File – Add SPARQL endpoint…*** menu. Click on any restricted role and try pressing ***F7 (Search PCA endpoint for WIP equivalent)*** and ***F4 (Search endpoints for URI)*** buttons. You will see some URIs found at PCA RDL (equivalent classes in *http://posccaesar.org/rdl/* will appear as search results) and some URIs found at iRING Sandbox.

You may find that some URIs are found in PCA RDL but without any meaningful data for them, and some URIs which will not be found anywhere. These are problems in reference data to be corrected in the future.

7.5. Organizing a project

Now let's go through building of a data project from several sources.

We'll use example piping system data prepared by Hans Teijgeler, one of the authors of ISO 15926 (see his personal website at http://www.15926.info/). Description of methodology used in preparation of this example can be found at http://www.15926.org/publications/general-discussions/pid-take-off/index.htm. Original files of the example can be downloaded from the page.

Example data uses template instances of the template set http://15926.org/15926_template_specs.php . This template set is developed by Hans Teijgeler and is currently reviewed by Special Interest Group Modeling, Methods and Technology of POSC Caesar Association. For use in the Editor environment templates can be downloaded from http://15926.org/15926_template_specs.php?mode=owl.

The process below is described for files downloaded at the moment of preparation of this guide. **The data at http://15926.org/15926_template_specs.php is work in progress and can be changed, augmented (and unfortunately sometimes corrupted) as data modelling goes on.** Templates are edited and corrected, and sometimes changes affect namespaces and encoding rules also.

To ensure compatibility with .15926 Editor patterns one version of this set is included with the distribution as ***templates.owl*** file in folder <*samples*>*pid* . If newer version in downloaded file looks or behaves differently – check for changes (especially in namespaces), or contacts us for help.

Start a new project (press *Ctrl*+*Shift*+N) and add template definition file to it. Drop ***templates.owl*** file on the Project panel.

Assign a module name to template definitions: click at the project node in Project panel, then find the field *Modules* in the Properties panel and click on it. Enter the key ***mmttpl*** and choose ***templates.owl*** from a dropdown menu.

Add a project reference data library file ***xyz-corp-rdl-extension.owl*** and open it in a data panel. Empty search in a *search* box will return 59 entities in the data tree. After expanding any entity's Properties group you'll find that identification of entities is done through *annUniqueName* annotation property.

Do the query **show(type = mmttpl.any)**. Three properly recognized template instances are returned, showing that template instances are indeed inked to template definition data source. opening them you'll see some missing roles and some missing entities in other roles indicating that the work on reference data is not completed.

It is also noticeable that this data source uses *owl:Class* declaration for all entities, which do not add any ISO 15926 specific semantic to the data. Also *rdfs:subClassOf* proprerty is used to record specialization relationship, which does not violate Part 8 requirements.

It is possible to find PCA RDL data for entities discovered in the project RDL. In local reference data you can find links to URIs of unrecognised entities in *http://rdl.rdlfacade.org/data#* namespace (marked with "?"). With the retirement of RDS/WIP reference data library only PCA

RDLand some sandboxes contain cross-references to these old URIs. Click an unrecognised URI and press *F7* button or use corresponding toolbar button. Special data view will be created with a result of PCA endpoint query for this URI. You can check the class and if necessary – drag it to the unrecognised property and update your reference data with new URI.

Now add the project data file ***pid-take-off.owl*** to the Editor. Do console queries:

show(type = part2.any.Thing)

show(type = mmttpl.any)

There are plenty of unnamed data entities, but query through *search* box reveals nothing. Look at any of 53 classes, expand *Patterns* node group and see that the new principle of identification is used here – entities are assigned identification through ClassifiedIdentificationOfClass template instance.

Query

show(type=mmttpl.ClassifiedIdentificationOfClass, valIdentifier=groupby, hasIdentified=out)

The list of identifiers comes out (each preceded with "**grouped by**" prefix) but entities behind them remain unrecognized – meaning that their modeling is still incomplete.

However you can see that template ClassifiedIdentificationOfClass has a hasUrClass role, meaning that each unrecognized class is a class created for partial OIM definition, and has a superclass representing the "true" piping system element, as described at http://www.15926.org/publications/general-discussions/class-model/index.htm . This allows us to form a query:

show(type = mmttpl.ClassifiedIdentificationOfClass, valIdentifier=groupby, hasUrClass=out)

and see the list of identifiers with entities available for exploration behind them

Through the *Patterns* node group you can now explore connections of entities in the model at UrClass level.

Notice that data is often incomplete, with some entities missing from the model. However links from project data to local reference data are recognized and can be navigated by *F12* button.

If you want to add entities to a data source – do not forget to set proper namespace for new entities (property grid of the data source is available when you click tree data tree root node). If you are owner of the data in the data source – you can use the namespace of existing entities. If you are going to enhance data prepared by others – use your own namespace for data you create.

You can organize and study another project from template set and data file found in *<samples>\iip* folder. This template set is a set used in iRING Tools, and data file is taken from Oil & Gas Interoperability (OGI) Phase 1 Pilot project guided by MIMOSA (http://iringug.org/wiki/index.php?title=GS_OGIDemo_001). The data file is an export from Bentley OpenPlant P&ID software.

***